The Dissertation Committee for Kihwan Jun Certifies
that this is the approved version of the following dissertation:

Improved Algorithms for Non-restoring Division and Square Root

Committee:

Earl E. Swartzlander, Jr., Supervisor

Mircea D. Driga

Dean P. Neikirk

Nur A. Touba

Michael J. Schulte

Improved Algorithms for Non-restoring Division and Square Root

by

Kihwan Jun, B.S., B.S.E.E., M.S.E., M.S.E.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

The University of Texas at Austin

December 2012

Dedication

To my parents, who have been there for me from day one and taught me what true love is. Thank you for all of the love, support, encouragement and dedication.

To my wife, who has been there for me for the last five years and showed me what true love is. Thank you for all of your love, support, patience, encouragement and dedication.

This is a tribute to the three of you.

Acknowledgements

I would like to thank all the people who have helped me accomplish this research.

First of all, I owe sincere and earnest thankfulness to my supervisor, Dr. Earl E. Swartzlander, Jr. for his invaluable guidance, support, and expertise in this field. He has guided me throughout my research with remarkable insight and profound knowledge. This dissertation would not have been possible without his guidance and support.

I am also grateful to the members of my dissertation committee, Dr. Mircea D. Driga, Dr. Dean P. Neikirk, Dr. Nur A. Touba and Dr. Michael J. Schulte. Their helpful advice on my research and their knowledge have guided me in my quest to do quality work.

I am obliged to many of my colleagues who supported me in the application specific processor group for their help. Especially, Inwook Kong and Jaehong Min spared me much of their time to helped me to give invaluable advice.

Finally, I would like to thank my wife, Hyungjoo Noh, who always supports and encourages me with great love, my parents, Myoungja Kim and Changhoon Jun, who gave me all their great wisdom throughout my life and mother-in-law, Songja Moon, for her love.

Improved Algorithms for Non-restoring Division and Square Root

Kihwan Jun, Ph.D.

The University of Texas at Austin, 2012

Supervisor: Earl E. Swartzlander, Jr.

This dissertation focuses on improving the non-restoring division and square root algorithm. Although the non-restoring division algorithm is the fastest and has less complexity among other radix-2 digit recurrence division algorithms, there are some possibilities to enhance its performance. To improve its performance, two new approaches are proposed here. In addition, the research scope is extended to seek an efficient algorithm for implementing non-restoring square root, which has similar steps to non-restoring division.

For the first proposed approach, the non-restoring divider with a modified algorithm is presented. The new algorithm changes the order of the flowchart, which reduces one unit delay of the multiplexer per every iteration. In addition, a new method to find a correct quotient is presented and it removes an error that the quotient is always odd number after a digit conversion from a digit converter from the quotient with digits 1 and -1 to conventional binary number.

The second proposed approach is a novel method to find a quotient bit for every iteration, which hides the total delay of the multiplexer with dual path calculation. The proposed method uses a Most Significant Carry (MSC) generator, which determines the sign of each remainder faster than the conventional carry lookahead adder and it

eventually reduces the total delay by almost 22% compared to the conventional non-restoring division algorithm.

Finally, an improved algorithm for non-restoring square root is proposed. The two concepts already applied to non-restoring division are adopted for improving the performance of a non-restoring square root since it has similar process to that of non-restoring division for finding square root. Additionally, a new method to find intermediate quotients is presented that removes an adder per an iteration to reduce the total area and power consumption. The non-restoring square root with MSC generator reduces total delay, area and power consumption significantly.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Computers have evolved rapidly since their creation. However, there is one thing that has not changed: The main purpose of computers is to do the arithmetic to run programs and applications. Basically, computers handle lots of numbers based on the three basic arithmetic operations of addition, multiplication and division. Compared to addition and multiplication, division is the least used operation. However, computers will experience performance degradation if division is ignored [1, 2, 3].

There are two kinds of division methods devised by researchers: digit recurrence division and division by convergence. Each method has its own advantages [1], however digit recurrence division which is simple and lower in complexity than division by convergence is the most common algorithm for division and square root in many floating point units [2, 4, 5, 6]. Restoring, non-restoring and SRT dividers are representative algorithms for digit recurrence division.

Although the non-restoring division algorithm is the fastest among the digit recurrence division methods (except for higher-radix SRT division) [7, 8], there are still a couple of things that can be modified to improve overall performance. First, for the non-restoring division algorithm, a partial remainder is necessary to find each quotient bit from each iteration and to determine which one among denominator and its complement is used for calculating next partial remainder along with current partial remainder. Total delay will be reduced if there are new methods to find a next partial remainder quickly or

if two tasks - finding each quotient and calculating next remainder - can be done simultaneously. Second, the non-restoring division algorithm consists of two major blocks per iteration and they are adders for calculating the partial remainder for the next stage and multiplexers for finding each quotient bit and determine operands. These two blocks are used n times if the non-restoring divider calculates an n-bit quotient. Therefore, the total delay, area and power consumption can be minimized if there is a novel approach to reduce the number of blocks or the number of bits handled by each iteration. Finally, the non-restoring division algorithm has a different quotient set, {-1, +1}, instead of conventional binary number set to increase the computation speed. Using the different quotient set reduces the delay of the non-restoring division and it only requires one addition per iteration where as the restoring division generally requires 1.5 additions per iteration [9]. However, there are a couple of drawbacks for using the different quotient set for the non-restoring division. First of all, it needs to convert its +1 and -1 quotient bits to a conventional binary number using an adder and a complement logic. Second, a least significant bit of a quotient for the non-restoring division is always odd number and it is critical error for the quotient [10]. If these drawbacks can be mitigated, the performance of the division by convergence algorithms will be improved.

1.2 RESEARCH DIRECTION

Although the non-restoring division algorithm has many advantages, there are possibilities to improve calculation speed and quotient accuracy. To improve the non-restoring division algorithm, three approaches are proposed in this dissertation.

The first approach is rearranging the order of the computation. By doing this, the delay of the multiplexer for selecting the quotient digit and determining the way to

calculate the partial remainder can be reduced by one unit delay per iteration. This reduction will rise as the number of bits are increased. This approach is similar to the way a carry select adder selects its intermediate results based on its carry [11]. It generates two possible intermediate sum results and carries simultaneously and then a multiplexer selects one of the two results once the carry reaches to the multiplexer. This approach reduces total delay significantly. Instead of generating two possible intermediate sum results, the modified non-restoring division algorithm generates two possible partial remainders. In addition, a new method to find a correct quotient will be discussed and it can further increase the quotient precision and reduce the elapsed time for finding the least significant bit of the quotient.

Second, the non-restoring division algorithm generates each quotient digit by checking whether the partial remainder is either positive or negative. So, the partial remainder is essential to find each quotient bit from each iteration and to determine which one among denominator and its complement is used for calculating next partial remainder along with current partial remainder. If there is an extra logic for determining the sign of each partial remainder faster than the conventional carry lookahead adder, the adder for calculating the partial remainder and the multiplexer are performed at the same time. Thus, the multiplexer delay can be totally ignored since the adder delay is generally longer than the multiplexer delay. A multilevel reverse most-significant-carry (MSC) computation algorithm is introduced to check the sign of the partial remainders [12]. The MSC generator produces the most significant carry faster than the conventional carry lookahead adder, provided the number of bits they calculate are the same. Therefore, the adder and the multiplexer with MSC generator performs simultaneously. In addition, a smaller adder with one bit less can be used since it doesn't have to generate the most significant bit for checking the sign of the partial remainders.

Finally, an improved algorithm for non-restoring square root is proposed. Two concepts already applied for non-restoring division above are adopted for improving the performance of non-restoring square root since it has a similar process. Additionally, a new method to find intermediate quotients is presented and it removes an adder per an iteration to reduce the total area and power consumption. Two new non-restoring square root algorithms are compared to each other with respect to the total delay, the area and total power consumption based on simulation result and it allows to choose the better algorithm among the two.

1.3 DISSERTATION ORGANIZATION

The proposed research focuses on improving the non-restoring division algorithms. The rest of this dissertation is organized as a total of five chapters. In Chapter 2, the conventional algorithm for digit recurrence division is explained with its detailed operation. The first approach, modified non-restoring division algorithm to reduce the total delay and improve its precision is presented in Chapter 3. Chapter 4 presents another approach to not only reduce the delay and improve its precision but also to occupy smaller area and consume less power than the approach presented in Chapter 3. The MSC generator, which achieves these goals, will be also explained. In Chapter 5,an improved algorithm for non-restoring square root is presented. Two approaches already presented in Chapter 3 and 4 are suitably modified for the non-restoring square root. They are compared to each other with respect to delay, area and power consumption for finding the best algorithm. Finally, a summary of the research work in this dissertation and suggestions for the future research are provided in Chapter 6.

Chapter 2

Background: Non-restoring Division

In this chapter, the concept of non-restoring division and the algorithm for its implementation is explained. The non-restoring division algorithm is a representative division algorithm among three major algorithms for digit recurrence division: restoring, non-restoring and SRT division. The digit recurrence division using subtractive division algorithm is most common in current microprocessors since the division-by-convergence algorithm has several drawbacks including necessity of a seed value lookup table and a multiplier even though the iteration has a speed advantage due to its quadratic convergence. [3, 7, 13]. Throughout this dissertation, the following notations are used for easy understanding.

Q: Quotient for the division

N: Numerator or Dividend

D: Denominator or Divisor

P_i: Partial remainder after the i-th iteration

TP_i: Temporary partial remainder after the i-th iteration

i: Number of iterations

n: Number of bits

q: Quotient set for the algorithm

It is assumed that all the numbers except i and n are binary numbers unless indicated otherwise. Q, N and D should be greater than 0 and D should be greater than N.

As presented here, the division algorithms are for fractions, but they can be easily modified so that they can calculate the significands for IEEE-754 floating-point numbers [14]. Floating-point numbers will be addressed in the next section which also explains how these numbers are dealt with in the division process [15].

2.1 FLOATING-POINT NUMBERS

Assuming the use of 32-bit two's complement integer number system, the range of the number system is from -2,147,483,648 to 2,147,483,647. Calculations cannot be performed when the numbers are either above the positive limit or below the negative limit. That means the range of numbers computers can handle are strictly limited. In addition, two's complement integer system cannot perform any arithmetic calculation associated with fractions. So, floating-point number systems are used to deal with a wider range of numbers for engineering and scientific computing.

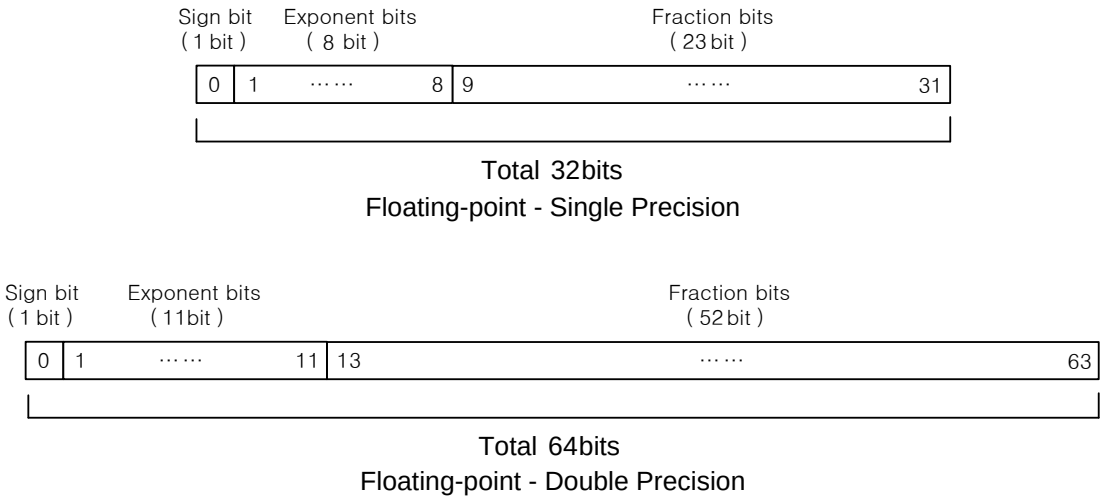


Figure 2.1: IEEE Standard 754 Floating-point number formats

Figure 2.1 shows the format of IEEE Standard 754 single and double precision floating-point numbers. Floating-point numbers consist of three components. First is the sign bit and it is always located in the most significant bit. (MSB) The number is positive if the sign bit is 0 and the number is negative if the sign bit is 1. Second, exponent bits are located between the sign bit and the fraction bits and they represent the exponent with a base of two. For single precision, there are eight exponent bits and the bias is 127. For double precision, there are 11 exponent bits and the bias is 1023. The last element is the fraction bits. The first bit among the fraction bits has a value of 0.5. The second fraction bit has a value of 0.25, etc. The fraction is converted to a significand by adding an integer one to the fraction bits. The significand is a mixed number in the range $1 \leq \text{significand} < 2$. To make it a fraction, the significand may be divided by two, so that it fits to perform division as either a numerator or a denominator. Figure 2.2 shows how to find the corresponding decimal number from the floating-point number.

$$\text{Significand} = 1 + \text{Fraction}$$

$$\text{Decimal value} = (-1)^{\text{Sign bit}} \times (2)^{\text{Exponent} - \text{Bias}} \times \text{Significand}$$

Figure 2.2: Floating-point to decimal conversion

Table 2.1 shows the minimum and maximum numbers that both single and double precision floating-point numbers can represent. Since exponent values of 0 and 255 for single precision and 0 and 1024 for double precision are reserved for other uses, the minimum and maximum values for the exponent are 1 and 254 for single precision and 1

and 1023 for double precision. Therefore, the floating-point number systems can deal with a wider range of values than a conventional fixed point number system.

Table 2.1: The range of floating-point number system

Precision	Exp (Min)	Exp (Max)	Bias	Minimum Number	Maximum Number
Single	1	254	127	$2^{1-127}\times(1.00\ldots0)$	$2^{254-127}\times(1.11\ldots1)$
Double	1	2046	1023	$2^{1-1023}\times(1.00\ldots0)$	$2^{2046-1023}\times(1.11\ldots1)$

2.2 RESTORING DIVISION

Before explaining the non-restoring division algorithm, the restoring division algorithm is briefly presented. The restoring division algorithm is the simplest of the three digit recurrence division methods and is similar to what is done by people with pencil and paper. A flowchart of the performing version of restoring division is shown on Figure 2.3. Once the restoring divider receives *N* and *D*, then the register of the partial remainder will store the value of *N* as its value. The next step is to calculate the temporary partial remainder, *TP*, which is obtained by subtracting *D* from *2P*. To obtain the quotient bit for this iteration, the temporary partial remainder is checked to see if it is greater than 0. If *TP* is greater than 0, the quotient digit is 1 and the value of *P* becomes the same as *TP*. Otherwise, the quotient digit will be 0 and *TP* needs to be "restored" by adding *D* to *TP* to get the correct value of *P*. This process requires one addition per bit if *TP* is greater than zero and two additions per bit if TP is less than zero.

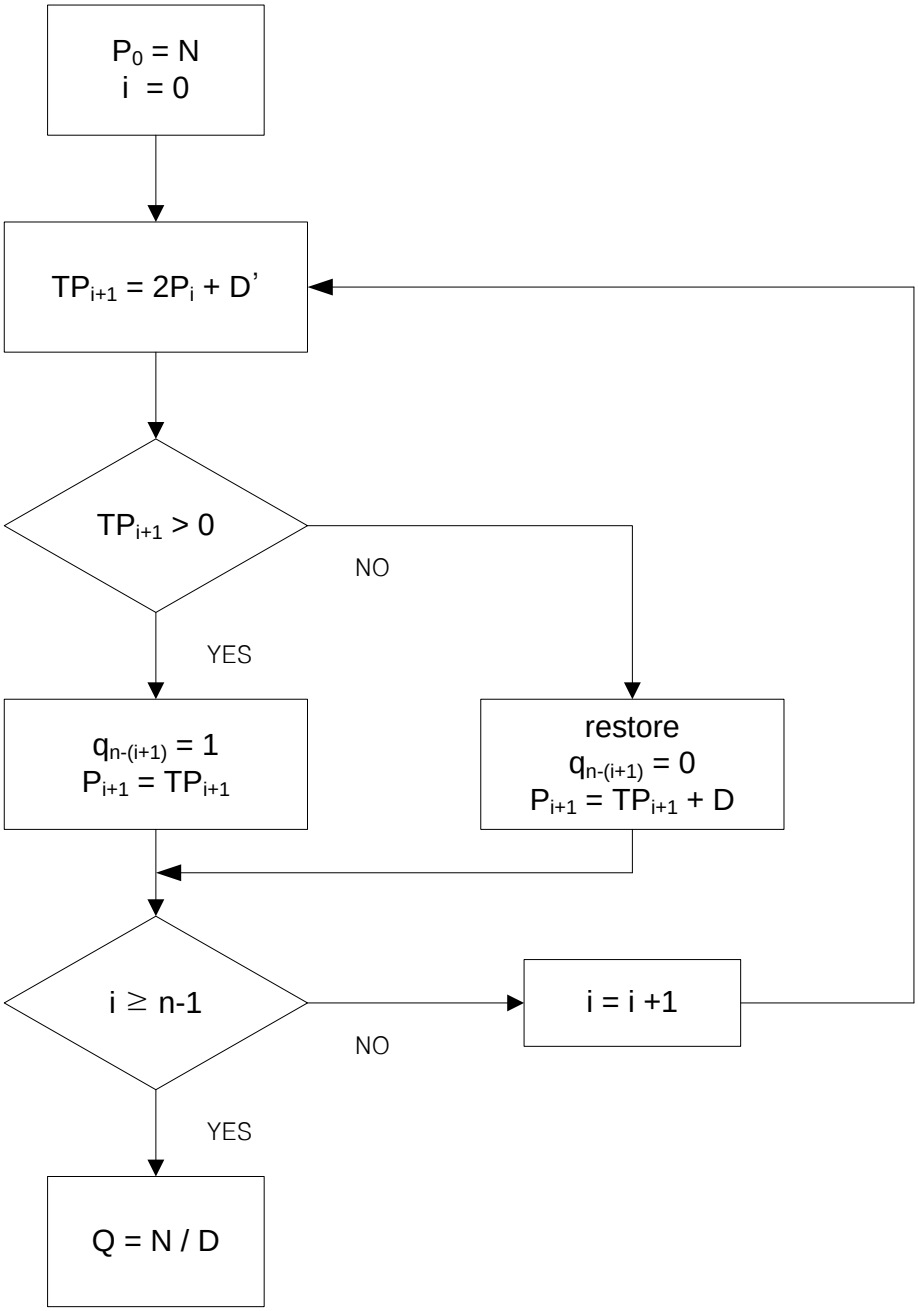


Figure 2.3: Flowchart of restoring division (performing version)

	Binary		Decimal
N:	0.1 0 1 0		5/8
D:	0.1 1 0 0		3/4
P_0:	0.1 0 1 0	$P_0 = N$	5/8
$2P_0$:	1.0 1 0 0	$TP_1 = 2P_0 - D$	5/4
D:	0.1 1 0 0		3/4
TP_1:	0.1 0 0 0	$TP_1 > 0 : q_3 = 1$	1/2
P_1:	0.1 0 0 0	$P_1 = TP_1$	1/2
$2P_1$:	1.0 0 0 0	$TP_2 = 2P_1 - D$	4/4
D:	0.1 1 0 0		3/4
TP_2:	0.0 1 0 0	$TP_2 > 0 : q_2 = 1$	1/4
P_2:	0.0 1 0 0	$P_2 = TP_2$	1/4
$2P_2$:	0.1 0 0 0		2/4
D:	0.1 1 0 0	$TP_3 = 2P_2 - D$	3/4
TP_3:	1.1 1 0 0	$TP_3 < 0 : q_1 = 0$	−1/4
D:	0.1 1 0 0		3/4
P_3:	0.1 0 0 0	$P_3 = TP_3 + D$	2/4
$2P_3$:	1.0 0 0 0		4/4
D:	0.1 1 0 0	$TP_4 = 2P_3 - D$	3/4
TP_4:	0.0 1 0 0	$TP_4 > 0 : q_0 = 1$	1/4
P_4:	0.1 0 0 0	$P_4 = TP_4$	2/4

Q = 0.1 1 0 1

Figure 2.4: Example of restoring division process (performing version)

Thus the average is 1.5 additions per iteration with 2 additions per iteration for the worst case. In order to reduce the delay, a modified form of restoring division, which is called 'Non-performing', has been devised and is explained in the next section.

2.3 NON-RESTORING DIVISION

For the performing version of restoring division, two additions are required for each iteration if the temporary partial remainder is less than zero and this results in making the worst case delay longer. To mitigate the delay during the iterations, the non-restoring division algorithm was devised [16]. It is shown by the flowchart of Figure 2.5. This algorithm does not use $TP_{i+1} = 2P_i - D$ process from the restoring division algorithm anymore. Instead, it only checks whether P is positive or negative based on the P-D plot shown Figure 2.6. When P is positive, then the quotient digit is one and the partial remainder is calculated by subtracting D from $2P$. Otherwise the quotient digit is negative and the partial remainder is increased by adding D to $2P$.

Compared to restoring division, non-restoring division has a different quotient set. While restoring division algorithm has zero and one as the quotient set, the quotient digits for non-restoring division are selected from the set {+1, -1}, with +1 corresponding to subtraction and -1 to addition. Using the different quotient set reduces the delay of non-restoring division algorithm compared to the performing version of restoring division algorithm. In other words, it only performs one addition per iteration which improves its arithmetic performance. However, the quotient with digit +1 and -1 for non-restoring division must be converted to a conventional binary number using an adder and complement logic as shown in Figure 2.7. This process requires three sub-steps and the first is to separate the quotient word that has both +1 and -1 bits into two intermediate

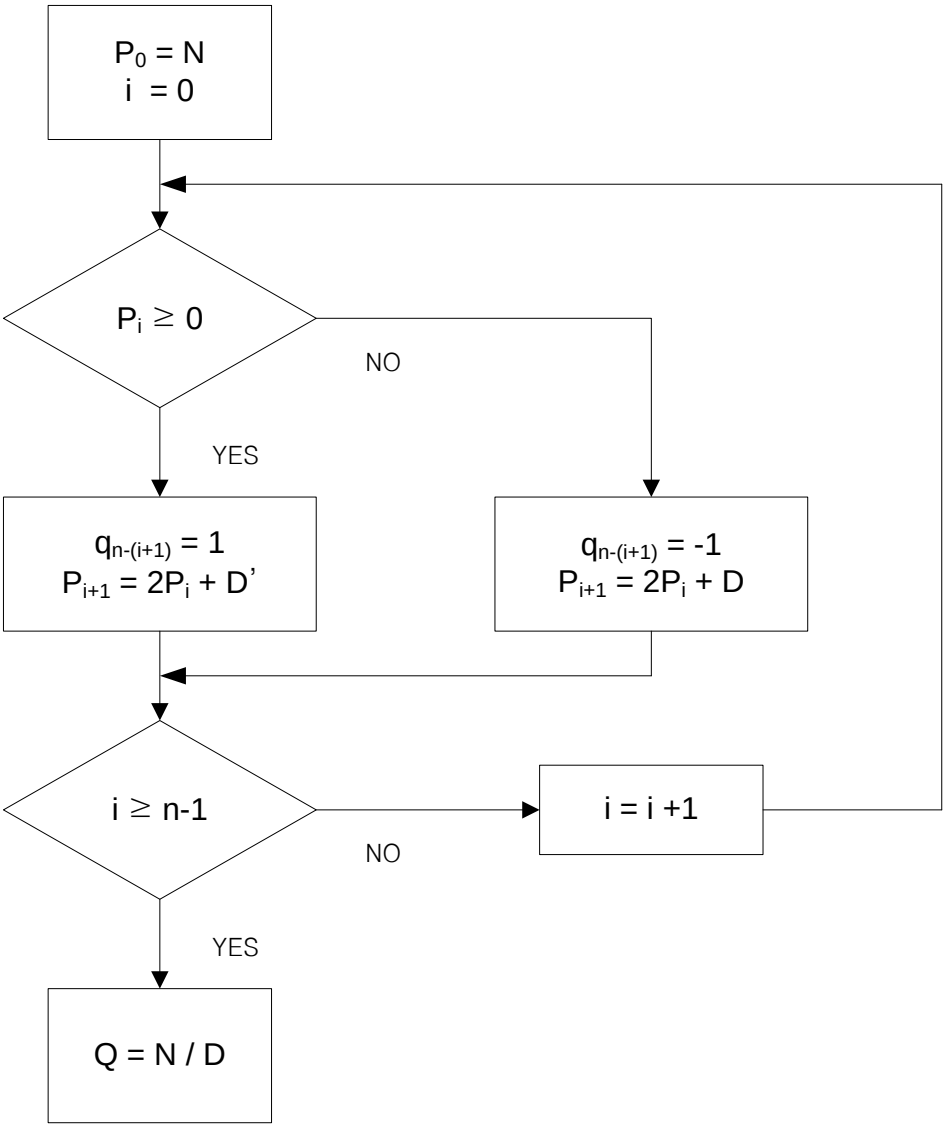


Figure 2.5: Flowchart of non-restoring division

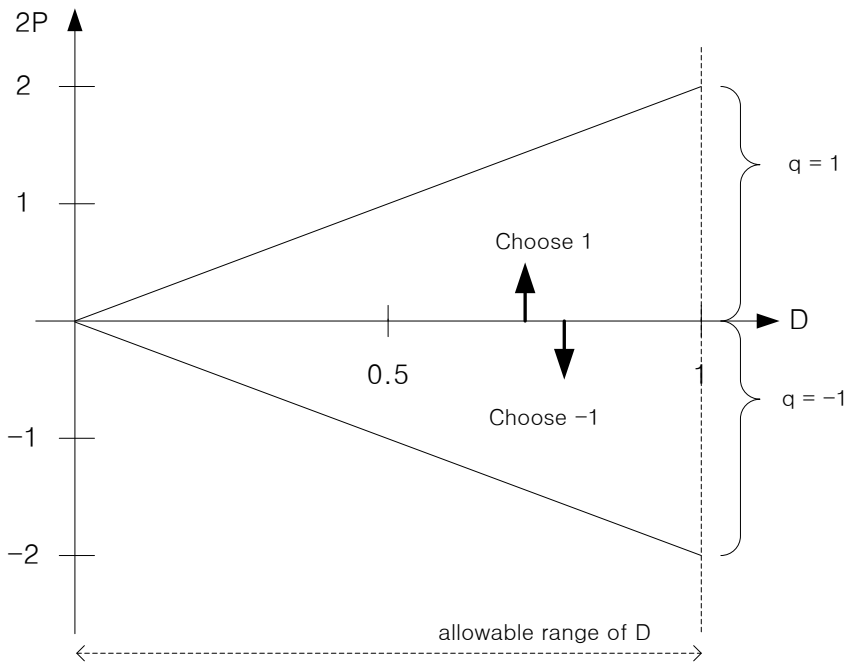


Figure 2.6: P-D plot for non-restoring division

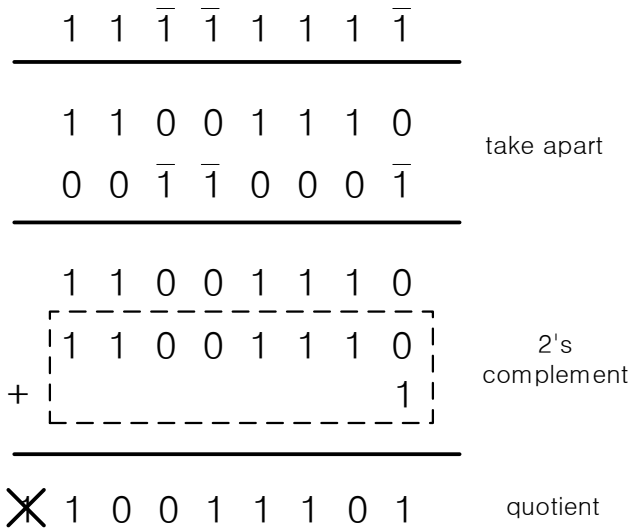


Figure 2.7: Example of converting a conventional unsigned binary number from the intermediate quotient result to a signed binary number

words, N and P. Word P consists of only +1 bits with zeroes that replace the -1 bits. While word N has 1 bits in place of the -1 bits and zeros that replace the +1 bits. Then, the N word is subtracted from the P word by forming the two's complement of the N word and adding it to the P word. In this process, the leading bit can be ignored since the resulting number an unsigned binary number. Figure 2.8 shows an example that illustrates how to find each quotient bit using non- restoring division.

As was already discussed in Chapter 1, there are a couple of ways to reduce the total delay by minimizing the delay per iteration. In addition, the result must be converted from the quotient with digits 1 and -1 to a conventional binary number. In Chapter 3 and 4, the proposed approaches address these issues to find a better algorithm with fast calculation, small area and low power consumption.

Binary		Decimal
N: 0.1 0 1 0		5/8
D: 0.1 1 0 0		3/4
P_0: 0.1 0 1 0	$P_0 = N$ $P_0 \geq 0 : q_4 = 1$	5/8
$2P_0$: 1.0 1 0 0		5/4
D: 0.1 1 0 0	$P_1 = 2P_0 - D$	3/4
P_1: 0.1 0 0 0		1/2
P_1: 0.1 0 0 0	$P_1 \geq 0 : q_3 = 1$	1/2
$2P_1$: 1.0 0 0 0		4/4
D: 0.1 1 0 0	$P_2 = 2P_1 - D$	3/4
P_2: 0.0 1 0 0		1/4
P_2: 0.0 1 0 0	$P_2 \geq 0 : q_2 = 1$	1/4
$2P_2$: 0.1 0 0 0		2/4
D: 0.1 1 0 0	$P_3 = 2P_2 - D$	3/4
P_3: 1.1 1 0 0		−1/4
P_3: 1.1 1 0 0	$P_3 < 0 : q_1 = -1$	−1/4
$2P_3$: 1.1 0 0 0		−2/4
D: 0.1 1 0 0	$P_4 = 2P_3 + D$	3/4
P_4: 0.0 1 0 0		1/4
P_4: 0.0 1 0 0	$P_4 \geq 0 : q_0 = 1$	1/4

$$Q = 0.1\ 1\ 1\ \overline{1}\ 1 = 0.1\ 1\ 0\ 1\ 1$$

Figure 2.8: Example of non-restoring division process

Chapter 3

Modified Non-restoring Division Algorithm

3.1 OVERVIEW

While the non-restoring division algorithm is the fastest and has less complexity among other radix-2 digit recurrence division algorithms [7], there are some possibilities to enhance its performance. To improve its performance, two new approaches are proposed here. The First approach is rearranging the order of the computation. The new algorithm changes the order of the flowchart, which reduces one unit delay of the multiplexer per iteration and this reduction will rise as the number of bits are increased. Secondly, a new method to find a correct quotient is presented and it removes an error that the quotient is always an odd number after the digit conversion from the quotient with digits 1 and -1 to a conventional binary number. Without another iteration for the error correction, the new logic to generate the LSB of the quotient quickly is also explained in this Chapter.

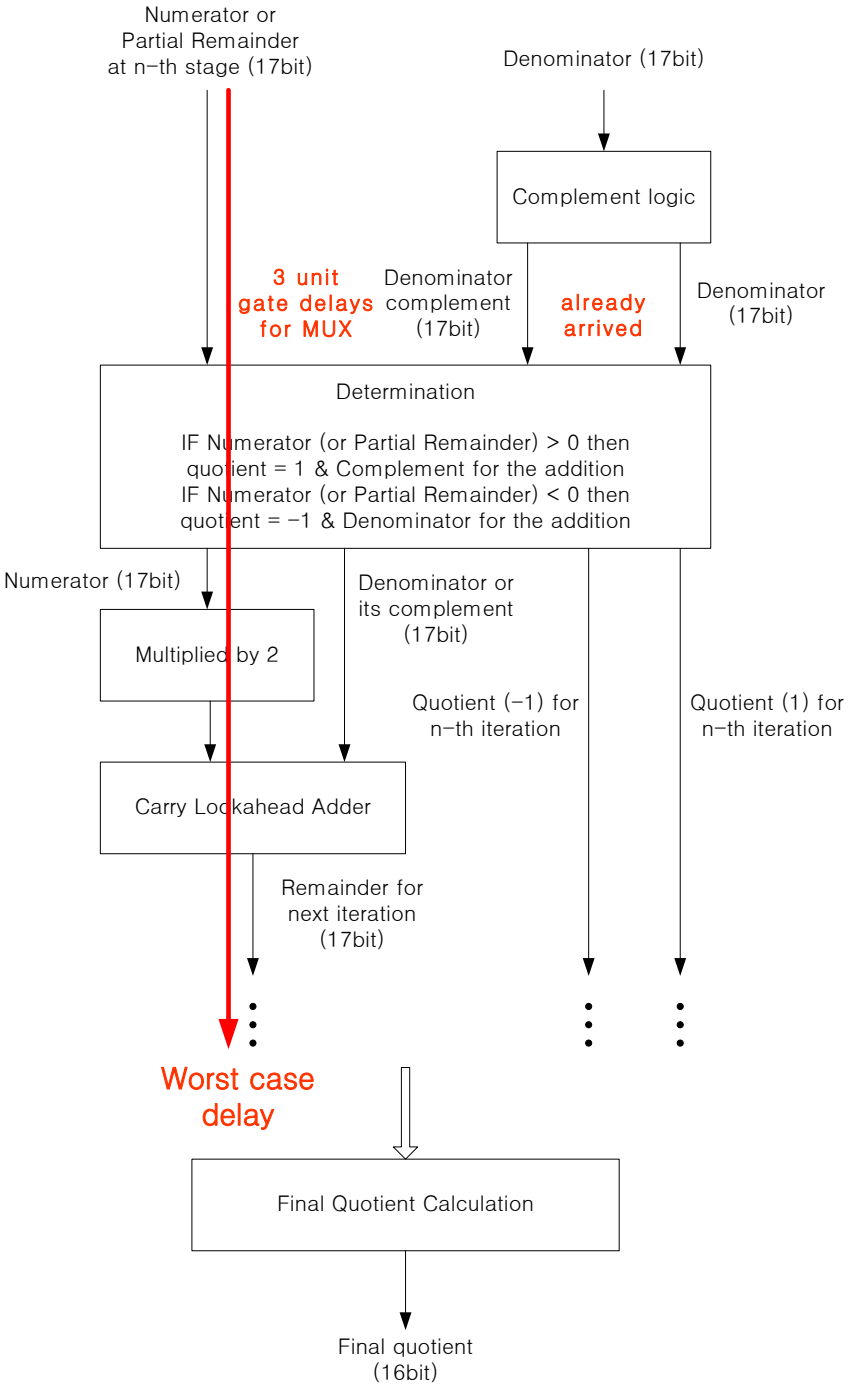
The first proposed approach is presented and it is compared with the conventional non-restoring division algorithm in Section 3.2. In Section 3.3, the new method to generate the LSB of the quotient using an MSC generator is explained and analyzed. Finally, the implementations and simulation results are discussed in Section 3.4. Please note that the simulation has been done with 8, 16 and 32-bit dividers although the significand of the floating point numbers are either 24 or 53 bits. By doing this, it is easy

to understand how the performance for each divider is varied as the number of bits increased and eventually the best divider can be selected with the various circumstances.

3.2 THE NEW ALGORITHM FOR REDUCING DELAY

Figure 3.1 shows the implementation of the standard non-restoring division algorithm for a 16-bit divider. When the numerator and denominator first enter the divider, the number of bits for both the numerator and denominator are extended from 16-bits to 17-bits to check their signs for either positive or negative. Then the complement logic changes denominator to one's complement form in the very beginning stage. After this process, the determination block (which consists of the multiplexer and one inverter) checks the sign of the numerator (at the first stage) or current partial remainder (after the first stage) and sets the quotient digit and determines which one among the denominator and its complement are to be used for calculating the next partial remainder along with the current partial remainder. Then the addition is performed to calculate the partial remainder for the next iteration. In this case, the worst case delay comes from the path starting at the determination block and ending at the carry lookahead adder.

The delay of a multiplexer can be reduced if the select signal reaches the multiplexer before the two input data arrive as shown in Figure 3.2. In the non-restoring division algorithm, the select signal is the most significant bit of the numerator or the current partial remainder. So, one unit of delay will be eliminated if the algorithm is modified to receive the numerator or the current partial remainder at the multiplexer before the two input data arrive there. To achieve this goal, the order of the determination block and the carry lookahead adders is switched as shown in Figure 3.3.



Non-Restoring Division for 16bit

Figure 3.1: Standard non-restoring division algorithm for a 16-bit divider

For the modified algorithm, the numerator or current partial remainder is at the determination block and the determination block selects the quotient digit before the adders calculate the possible partial remainders. So, the multiplexer delay can be reduced from three unit of delay to two.

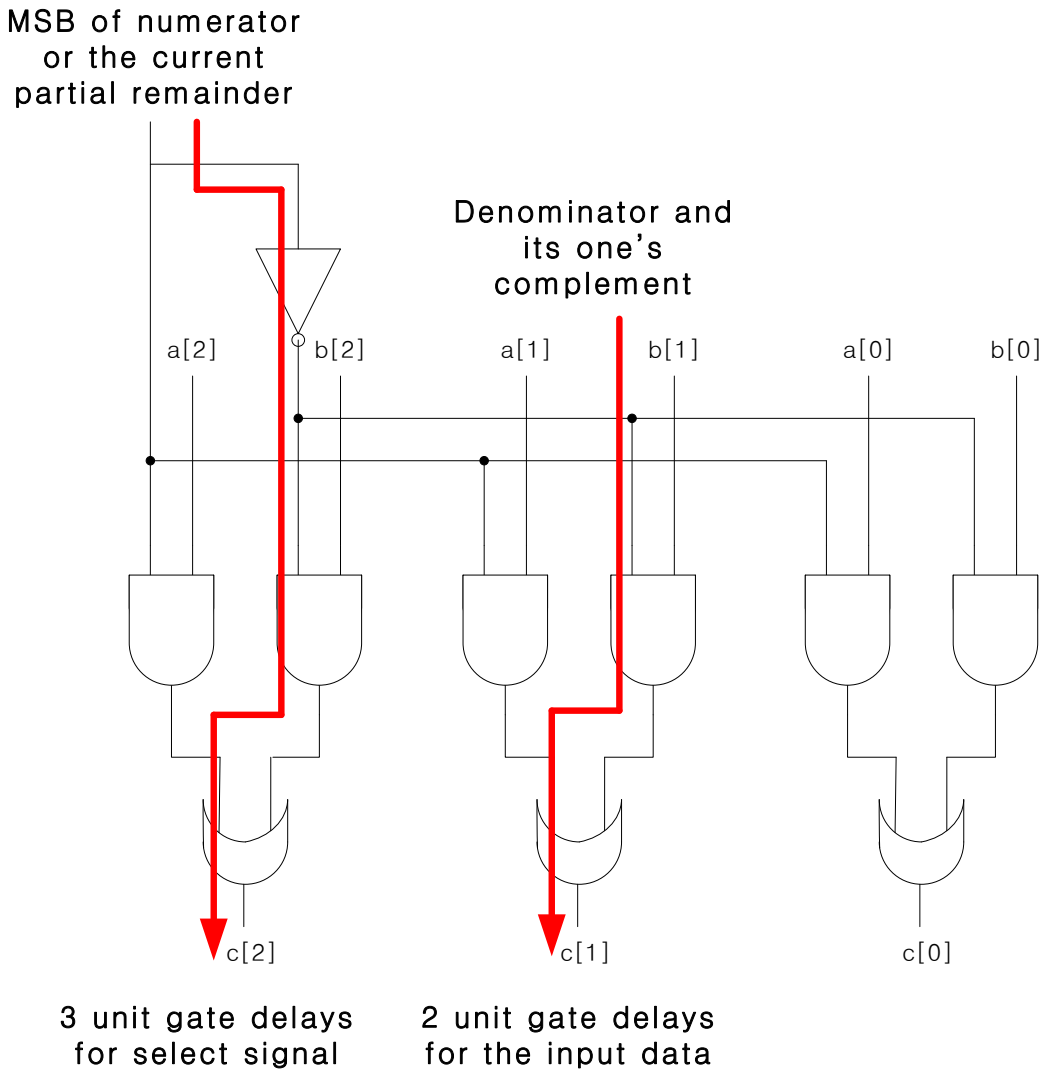
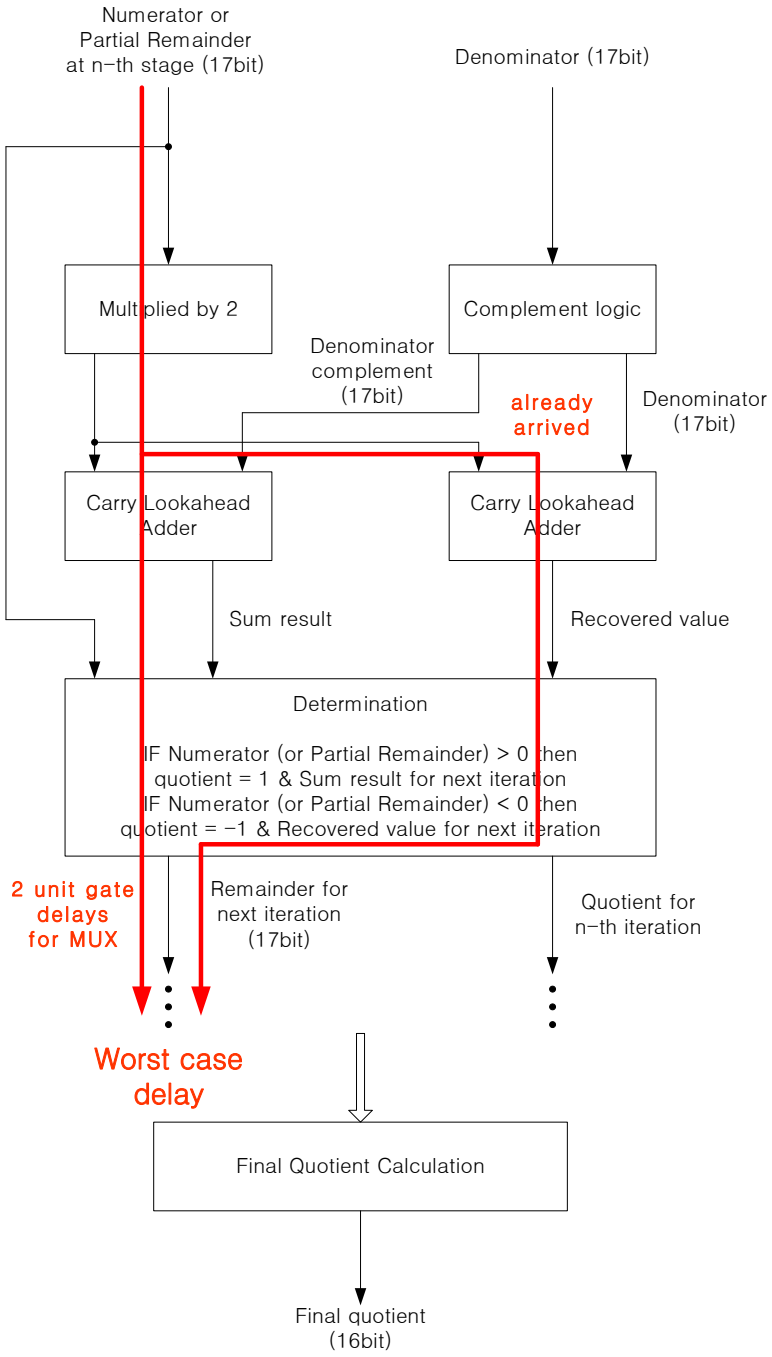


Figure 3.2: The delay comparison between the select signal and input data at the multiplexer



Modified Non-Restoring Division for 16bit

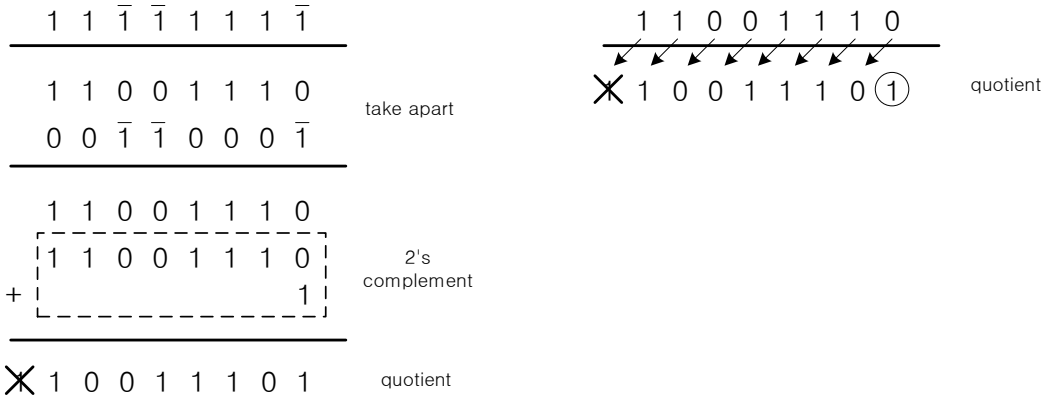
Figure 3.3: Modified non-restoring division algorithm for a 16-bit divider

Although this delay reduction is relatively small, n iterations means it can critically affect the overall delay profile. Doubling the number of the adders is tradeoff for reducing the delay. A similar structure is used for a carry select adder [10, 11]. This adder generates two possible results in advance for final selection by the carry-in from its previous adder. For the two results, one is based on a carry-in of one and the other is based on a carry-in of zero. Once the carry is known, then a multiplexer selects the correct result. The delay is reduced.

3.3 NOVEL METHOD TO CORRECT THE QUOTIENT ERROR

The non-restoring division algorithm has a different quotient set instead of the conventional binary number set to increase the computation speed. Using the different quotient set reduces the delay of non-restoring division and it only requires one addition per iteration where as the restoring division generally requires 1.5 additions per iteration. However, there are a couple of drawbacks for using the different quotient set for the non-restoring division. First, the quotient with digit +1 and -1 for non-restoring division must be converted to a conventional binary number using a digit conversion logic. On-the-fly conversion is commonly used and replaces the conventional digit converter as shown in Figure 3.4(a) with 1-bit left shifter and one register with signed binary number system as shown in Figure 3.4(b) [10,13]. There still is a problem regarding the least significant bit of the quotient for the non-restoring division algorithm although a digit converter generates the final quotient. The least significant bit of a quotient for the non-restoring division always sets to one regardless its next iteration since 2's complement of its -1 quotient word is used as a part of a conversion logic. It generates an error of 2^{-n} for an n-bit divider. To eliminate this problem, a MSC generator to generate the correct LSB of

the quotient quickly is explained in Section 3.3.2. Second, one multiplexer for selecting the first quotient, q_0, can be removed since it does not affect the final quotient. So, the delay from one multiplexer will be eliminated and it is presented in Section 3.3.1.



(a) conventional conversion (b) on-the-fly conversion

Figure 3.4: Two method for converting from the set of digit +1 and -1 to a conventional binary number

3.3.1 Algorithm Simplification

There is one additional process needed to convert the intermediate quotient with positive and negative one digits into a conventional binary number. This process requires 2 sub-process. This process requires three sub-steps and the first is to separate the quotient word mixed both +1 and -1 bits into two quotient words. One quotient word as a result of the first step, P, consist of only the +1 bits with zeroes that replace the -1 bits and the other word, N, consists of +1 bits in place of the -1 bits with zeros that replace the

+1 bits. Then, N is subtracted from P by forming the two's complement of N and adding it to P, which produces a conventional binary number. In this process, the leading bit can be ignored since the resulting number is an unsigned binary number. This includes a complement and an addition sub-process as shown in Figure 3.4(a).

The one's complement of N is exactly the same as P. The result after the addition is the same as a 1-bit shift left of P. A one is inserted at the LSB of the final quotient in all cases to convert the one's complement of N to its two's complement. By adopting this scheme, the steps of computing the one's complement and the addition are removed from the non-restoring division algorithm and it allows the quotient to be calculated faster.

As discussed above, one multiplexer for selecting the first quotient, q_0, can be removed since it does not affect the final quotient. So, the delay from one multiplexer will be eliminated. Since the numerator entering into the floating-point divider is always positive [14], the leading bit can be ignored and one-bit quotient generated at the first stage is not required for the final quotient as shown in Figure 3.4, only the partial remainder is needed for the second stage. Therefore, the determination block consists of one multiplexer at the first stage can be eliminated for algorithm simplification. This will also increase the calculation speed.

3.3.2 The Most Significant Carry (MSC) Generator

The least significant bit of a quotient for the non-restoring division always sets to one regardless its next iteration as shown in Figure 3.5 since the 2's complement of N is used as a part of a conversion logic. It generates an error of 2^{-n} for n-bit divider. To remove the error, the new logic to the generate correct LSB of the quotient quickly is explained in this section.

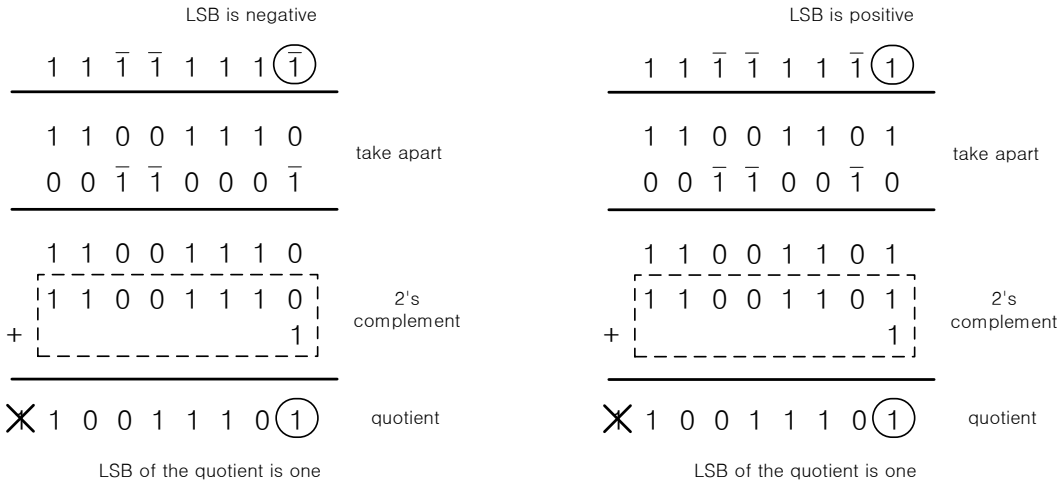


Figure 3.5: The least significant bit for each final quotient in both cases

Figure 3.6 shows the final stage for the 16-bit modified non-restoring division algorithm with the proposed approach as shown in Section 3.2. The least significant bit of the intermediate quotient, quotient[0], is generated at the final determination block from the partial remainder generated the previous iteration. Since the quotient[0] will be the second least significant bit for the final quotient after the on-the-fly digit converter, there are no more iterations to generate the least significant bit for the final quotient. Once the 16-bit intermediate quotient is calculated, the final quotient is obtained by on-the-fly digit converter, shifts the intermediate quotient by 1-bit left of and inserts a one at the LSB of the final quotient. During the conversion process, the least significant bit of final quotient always sets to one regardless any circumstances as shown in Figure 3.7 (a).

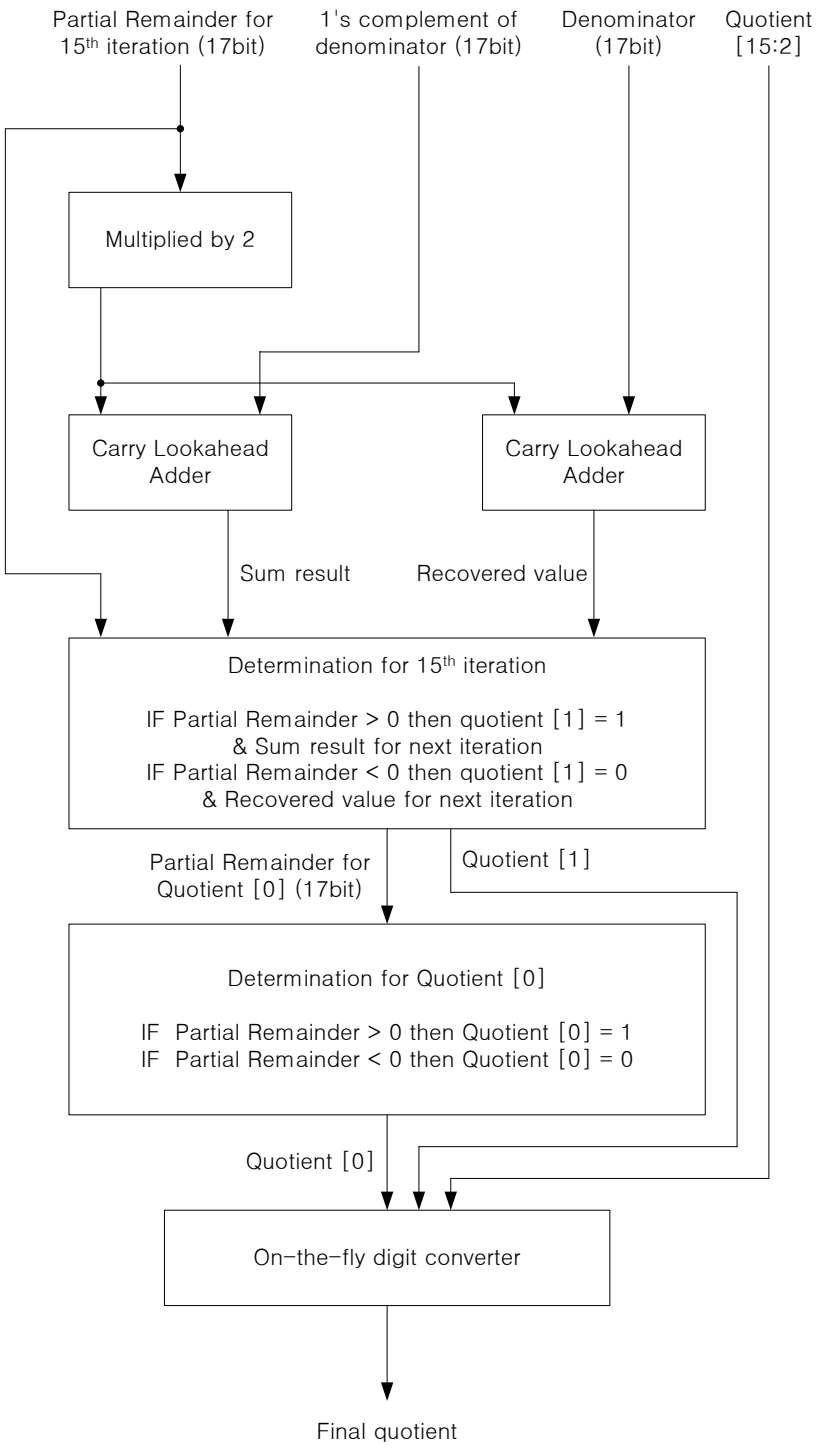
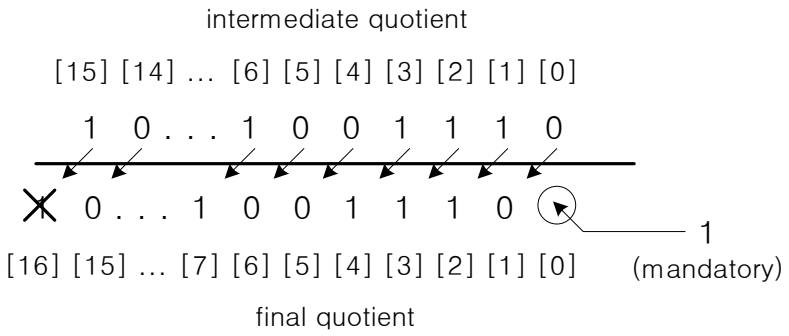
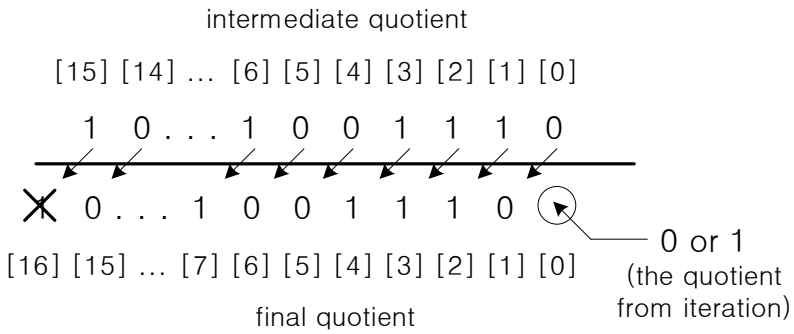


Figure 3.6: The final stage for the 16-bit modified non-restoring division algorithm with possible quotient error



(a) algorithm with possible quotient error



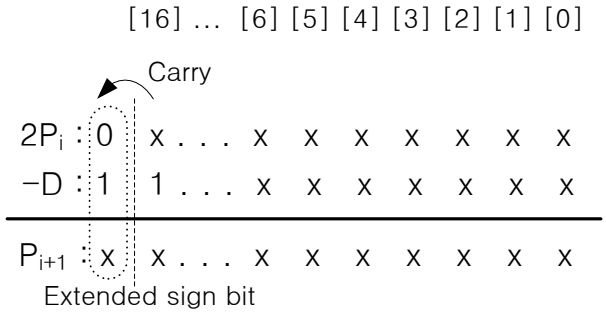
(b) algorithm without possible quotient error

Figure 3.7: Required iteration for eliminating possible quotient error

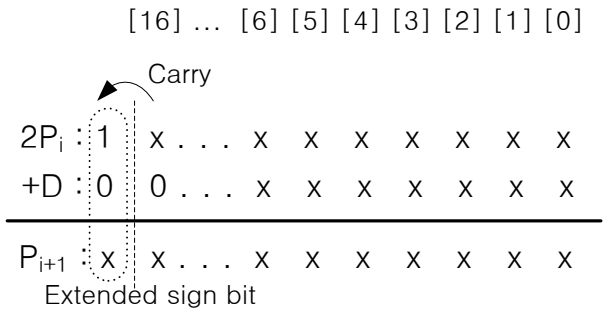
To enhance the accuracy for the least significant bit of the final quotient, an extra iteration is needed and it contains two carry lookahead adders and one inverter [10]. The simplest way to achieve this goal is do the one full iteration after the 15th iteration to generate another partial remainder for the least significant bit for the final quotient. Therefore, an additional iteration process including two 17-bit carry lookahead adders, a multiplexer and an inverter should be added for correct the possible quotient error. An inverter is used for determining the least significant bit of the final quotient since inverting the most significant number of the final partial remainder is the same as the

final quotient. If the partial remainder is less than zero, its most significant bit is one. Once inverting it through an inverter, the result is zero and it becomes the least significant bit of the final quotient.

The total area and the worst case delay resulting from the extra iteration is increased considerably. The whole iteration is not required if the most significant bit of the last partial remainder is known. Since the least significant bit of the final quotient is directly obtained from the most significant bit of the last partial remainder, the exact value of the partial remainder is also not necessary. If there is a faster and simpler alternative to find whether the partial remainder is positive or negative, then the extra iteration is no longer required.



(a) Partial remainder is positive



(b) Partial remainder is negative

Figure 3.8: A new method to check the sign of the partial remainder

Figure 3.8 shows how to check the most significant bit of the partial remainder. Initially, both the numerator and the denominator are all positive since they are come from the significands in the floating-point numbers and the significands may be divided by two, so that it fits to perform division as either a numerator or a denominator [14].

Based on the non-restoring algorithm, the positive current remainder is always paired with a negative denominator, the ones complement of the denominator, to calculate the next partial remainder. On the other hand, the negative current remainder is always paired with a positive denominator to calculate the next partial remainder as shown in Figure 3.8. In other words, the most significant bit of the denominator, +D, is always zero if the most significant bit of the partial remainder, P_i, is one. And, the most significant bit of the two's complement of the denominator, -D, is always one if the most significant bit of the partial remainder, P_i, is zero. Since the extended sign bit of the next partial remainder, P_{i+1}, decides whether it is greater than zero or less than zero, the carry resulting from the addition at n+1-th bit is one, then the extended sign bit of the next partial remainder, P_{i+1}, is set to zero and the final quotient is generated as one. If the carry is not propagated to the extended sign bit, the partial remainder is assumed to be less than zero and the last quotient is zero.

A Multilevel reverse most-significant-carry computation algorithm is used to calculate the most significant carry (MSC) quickly [12, 17, 18, 19]. It was originally devised for compound adders to compute the addition either with or without its carry, so early determination of the carry reduces the overall delay. The basic concept of the MSC determination is as follows. First, check the carry propagation chain from the MSB of both operands by using exclusive-OR gates. Then, check the MSC by using AND gates of both operands where the carry propagation chain is lost. Figure 3.9 shows an example of determining the MSC.

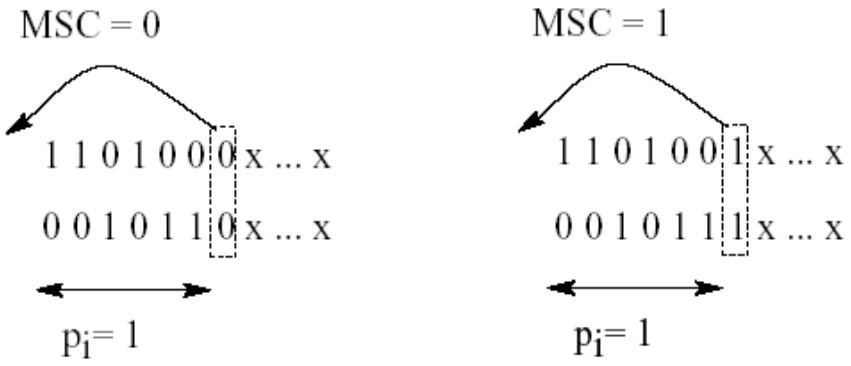


Figure 3.9: Example of MSC determination

This also can be done using both AND and OR gates without the Exclusive-OR gates. The algorithm for the most significant carry (MSC) generator modified for non-restoring division is shown in Figure 3.10. The intermediate result g_i is generated by an AND operation of both operands and k_i is also generated by an OR operation. Then, h_i, the MSC chain, is generated by an AND operation from the most significant bit to each of the remaining bits in order. Using previously generated g_{i+1} and h_i make d_i successively. Finally, OR operations from the most significant bit to each of the remaining bits in order can generate the MSC. Figure 3.11 shows how the non-restoring division algorithm with the MSC generator is implemented to enhance the accuracy of the final quotient.

For verification purpose, the estimated delays and complexities for each error correction method is presented in Table 3.1 and 3.2. Note that one iteration includes two carry lookahead adders, a multiplexer and a inverter as shown in Figure 3.6. The error correction method with the MSC generator counts a multiplexer and an MSC generator as shown in Figure 3.11. Also note that either a 2-input AND or OR gate or an inverter is

counted as 1 gate. From the delay perspective, either a 2-input AND or OR gate or an inverter has 1Δ delay.

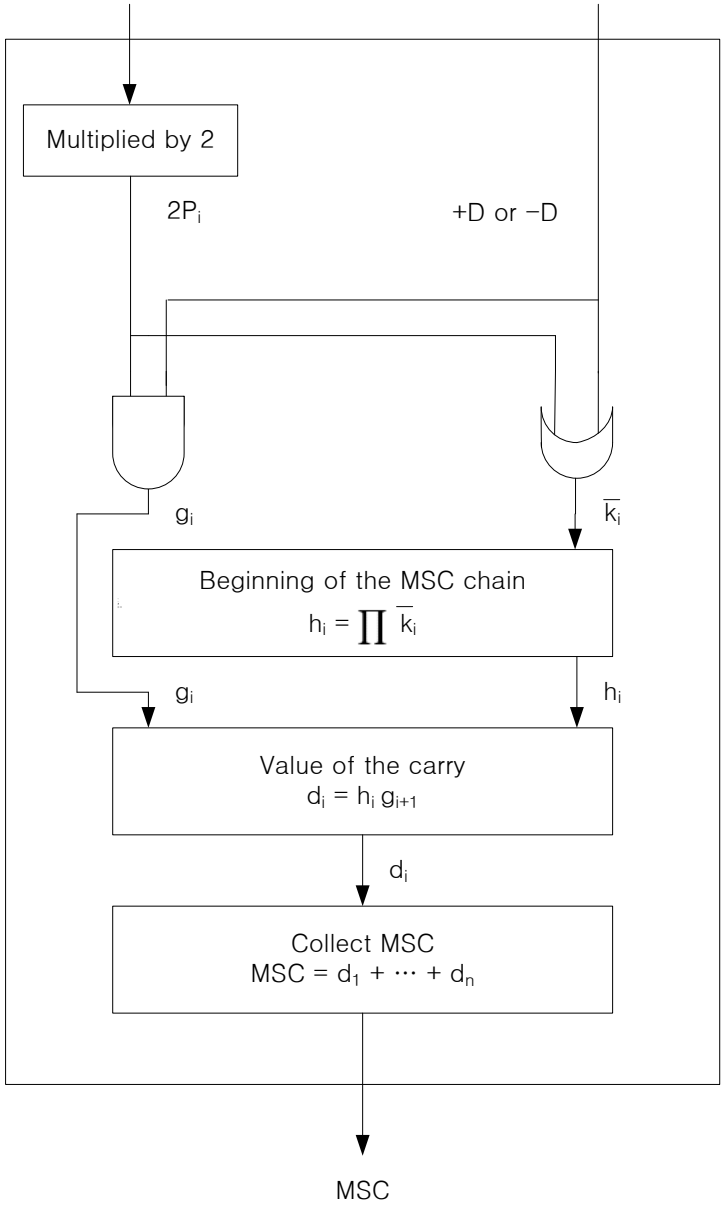


Figure 3.10: Most significant carry (MSC) generator for the 16-bit non-restoring division

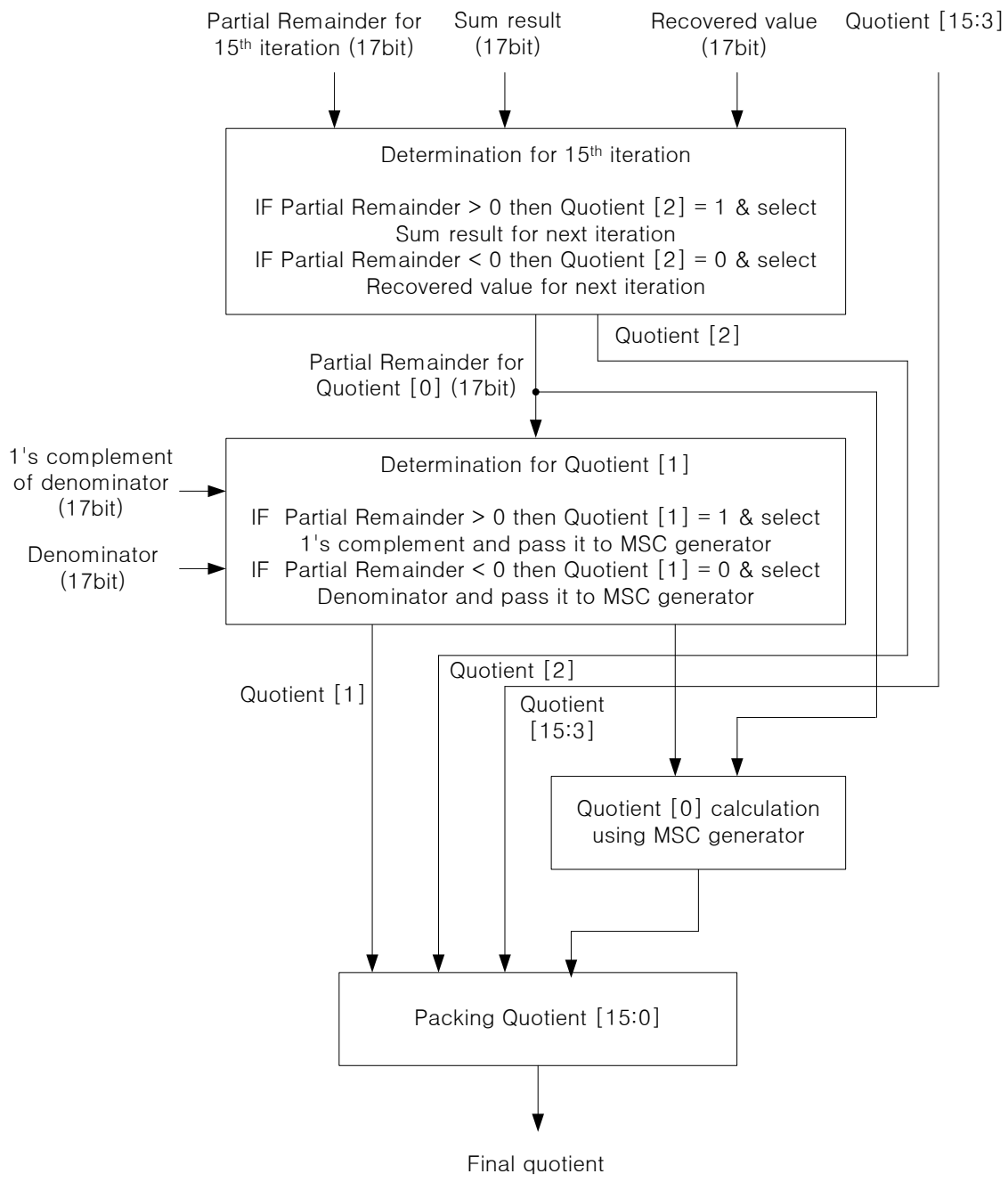


Figure 3.11: The final stage for the 16-bit modified non-restoring division algorithm with the MSC generator

Table 3.1: Estimated worst case delays for each error correction methods

Worst case delay	One iteration	MSC generator	Difference
9 bit	15Δ	11Δ	4Δ (-26.7%)
17 bit	17Δ	12Δ	5Δ (-29.4%)
33 bit	21Δ	13Δ	8Δ (-38.1%)

Table 3.2: Complexities for each error correction methods

Complexity	One iteration	MSC generator	Difference
9 bit	277	71	224 (-74.4%)
17 bit	551	137	414 (-75.1%)
33 bit	1085	272	879 (-74.9%)

Table 3.1 shows that the estimated worst case delays for the method with the MSC generator is almost 38% less than its delay per iteration for 32-bit division algorithm and it is evident that the MSC calculation algorithm is much faster than the method with carry lookahead adder. Table 3.2 also shows the number of the logic gates used for the proposed method is much less than conventional iteration method.

A new algorithm for correcting the least significant bit of a quotient for the non-restoring division has been presented. The modified algorithm increases its speed and reduces the complexity of the MSC generator. It has been analyzed that the proposed method to find the last quotient digit reduces the total delay by almost 38% per iteration.

The reduction is greater as the word size increases. The total area is also decreased by over 74%. In Section 3.4, the new algorithm will be verified for correctness using Verilog models.

3.4 VERIFICATION AND SIMULATION RESULTS

In this section, simulations are performed for verification purposes. Nine different logic circuits are implemented using the Verilog Hardware Description Language (Verilog HDL). The FreePDK45nm version 1.0 is used as a library file for the delay, the area and the power consumption. First, 8, 16 and 32-bit dividers using the standard non-restoring division algorithm that corrects the error by one more iteration are implemented and simulated. Then, the modified non-restoring dividers using either the MSC generator or the carry lookahead adder for error correction with 8, 16 and 32-bits are also designed and simulated. To find the delays, the Synopsys Verilog Compiler Simulator (VCS) program is used to run the simulation with various Verilog HDL files for the dividers. Design Vision is also used for finding the area and the power consumption. A hundred random vectors are generated as both the numerators and the denominators for each simulation and the delays in Table 3.3 are the average values of a hundred simulations.

Table 3.3: Average delays for each division algorithm

delay	Standard NRD (a)	Modified NRD with CLA (b)	Modified NRD with MSC (c)	Diff. (a-b)	Diff. (a-c)
8 bit	21.04 ns	17.38 ns	16.66 ns	3.66 ns	4.38 ns
16 bit	51.17 ns	41.30 ns	40.30 ns	9.87 ns	10.87 ns
32 bit	112.22 ns	89.71 ns	88.29 ns	22.51 ns	23.93 ns

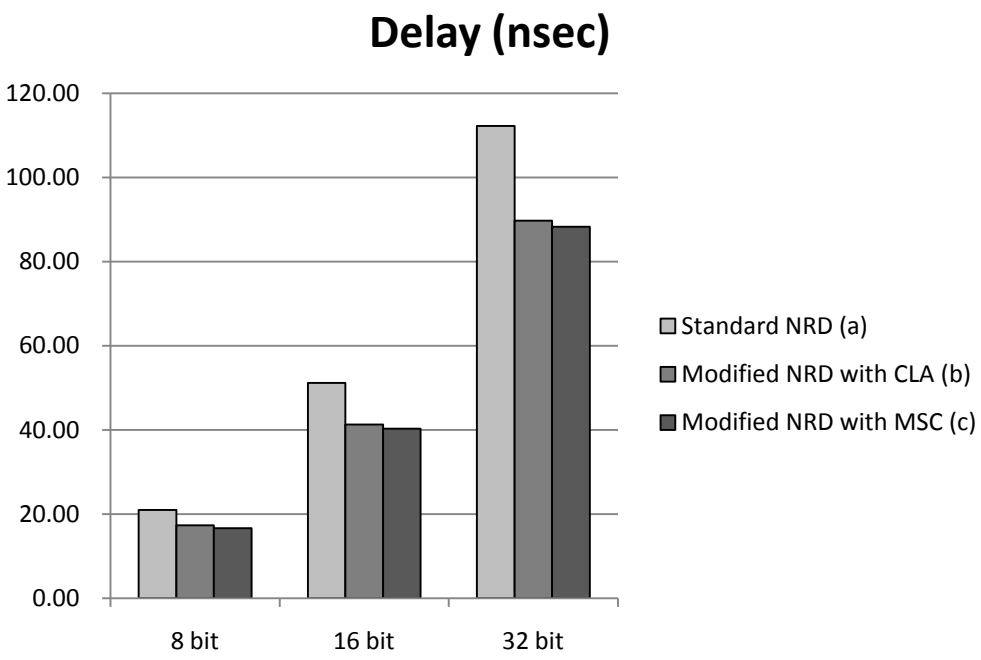


Figure 3.12: Graph of the delays for each division algorithm

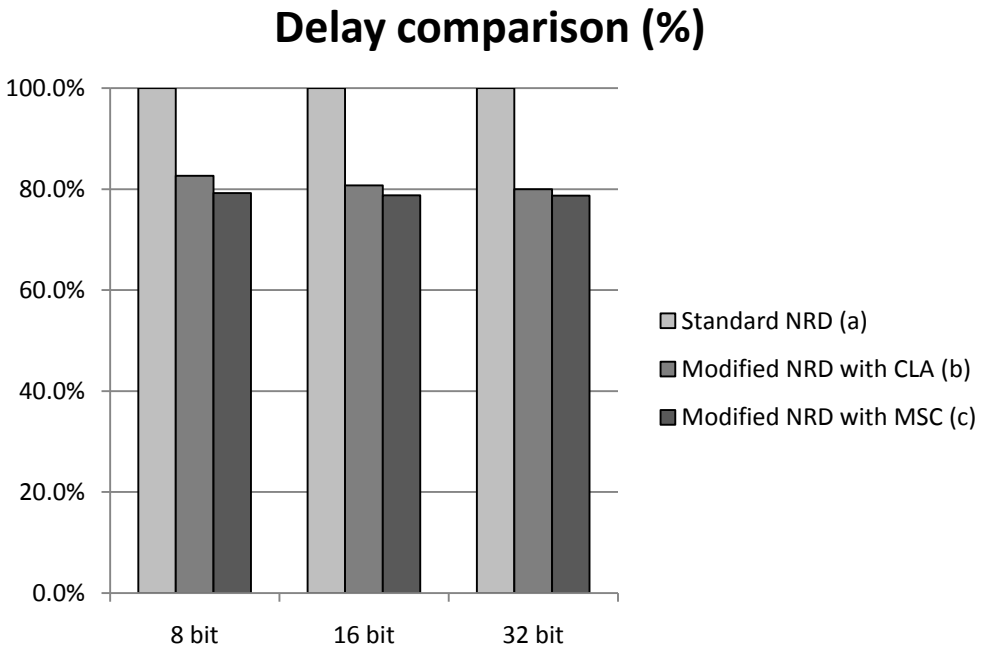


Figure 3.13: Graph of the delay comparison between the three algorithms

Based on Table 3.3, the modified non-restoring division algorithm with the MSC generator is the fastest and dramatically reduces the total delays compared to the standard non-restoring division algorithm. The modified non-restoring division algorithm with the MSC generator reduces the delay by up to 1.4ns compared to the modified non-restoring division algorithm with a CLA, the MSC generator reduces the delay up to 1.4ns. For 8-bit comparison, the modified non-restoring division algorithm with the MSC generator has a delay of 16.66 nanoseconds, which is almost 21% less than the standard. For 16-bit and 32-bit dividers, the delays are reduced by 21.2% and 21.3% respectively. As shown in Figure 3.12, the reduction in the total delays becomes slightly larger as the number of bits is increased and switching the order of the algorithm is more effective to minimize the total delay in the larger dividers.

Table 3.4: Delay ranges for each division algorithm (nsec)

Delay Range	Standard NRD			Modified NRD with CLA			Modified NRD with MSC		
	Min	Max	Diff	Min	Max	Diff	Min	Max	Diff
8 bit	15.27	27.06	11.79	12.82	20.33	7.51	13.38	20.55	7.17
16 bit	36.74	61.88	25.14	32.63	51.39	18.76	27.40	49.66	22.26
32 bit	90.72	130.67	39.95	75.85	102.76	26.91	72.53	99.98	27.45

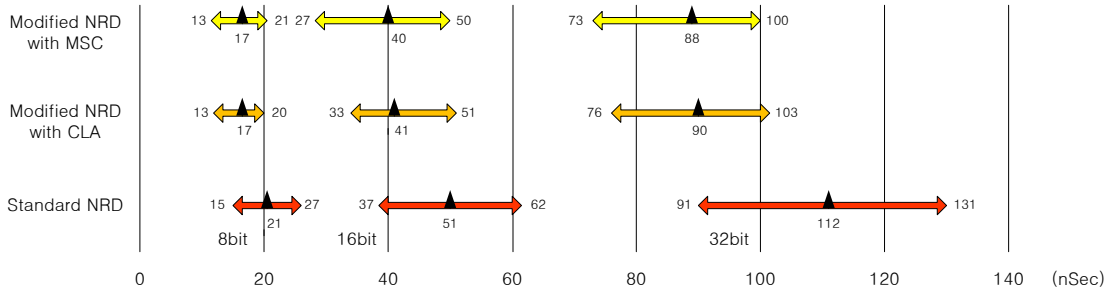


Figure 3.14: Delay ranges for the division algorithms

Depending on the vectors, the delays vary over a fairly wide range. Table 3.4 shows both minimum and maximum delays for each division algorithm. The two modified non-restoring division algorithms have a narrower range than the standard non-restoring division and indicates the two former algorithms are more stable than the latter one. That is the deviations from the average for the modified non-restoring divisions are smaller than for the standard non-restoring division. The range of the delays for the modified non-restoring divisions is reduce as the word size is increased. Comparing the two modified non-restoring division algorithms with each other, there are only small differences. Figure 3.14 also shows the overall trend of the delay range as the word size is increased

Since the modified non-restoring division algorithms have almost twice as many adders as the standard non-restoring divider, it occupies more area than the standard divider even though some logic is removed as shown in Table 3.5. The 8-bit standard non-restoring division algorithm has an area of 2590μm^2 where as the modified one with the MSC generator occupies 3999μm^2 which is almost a 54% increase as shown in Figure 3.16. For 16-bit and 32-bit dividers, the areas are increased by 70.6% and 78.7%, respectively.

Table 3.5: Areas for each division algorithm

Area	Standard NRD (a)	Modified NRD with CLA (b)	Modified NRD with MSC (c)	Diff. (a-b)	Diff. (a-c)
8 bit	2590 μm^2	4460 μm^2	3999 μm^2	1870 μm^2	1409 μm^2
16 bit	10290 μm^2	18486 μm^2	17556 μm^2	8196 μm^2	7266 μm^2
32 bit	40806 μm^2	74793 μm^2	72945 μm^2	33987 μm^2	32139 μm^2

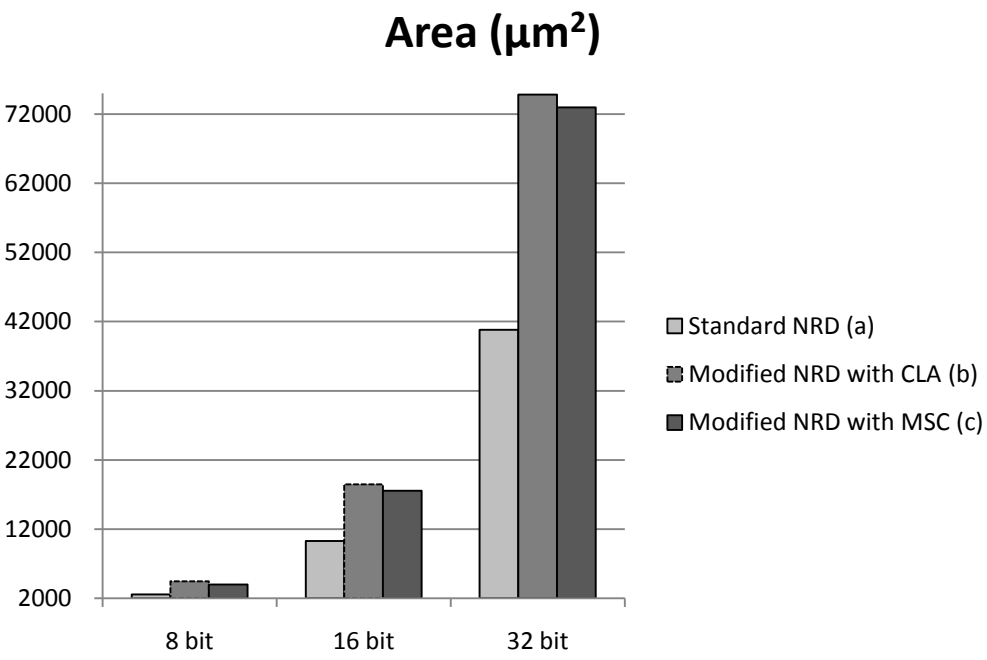


Figure 3.15: Graph of the areas for each division algorithm

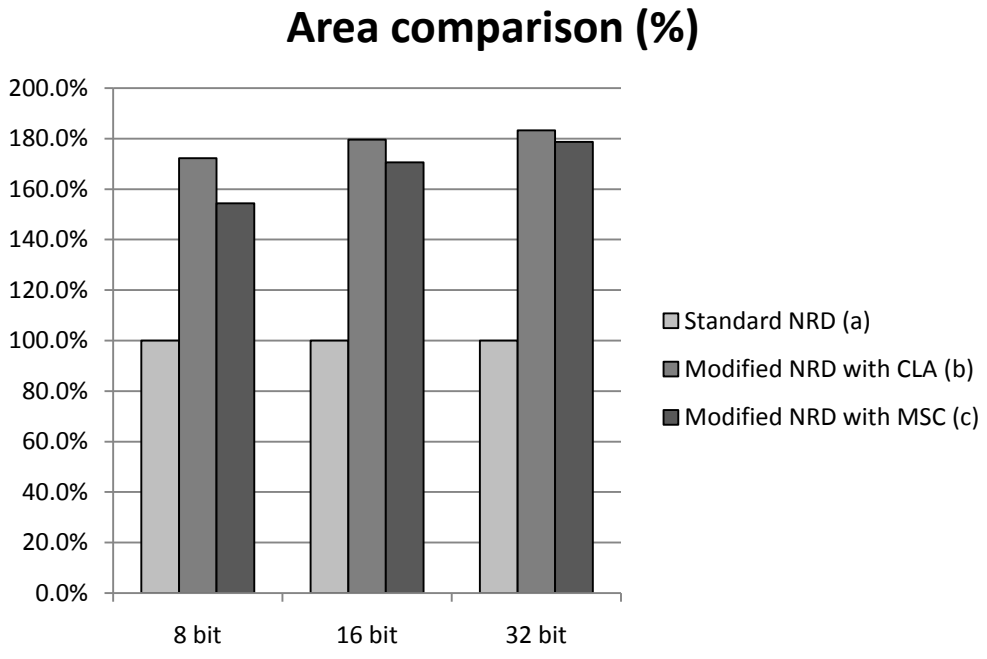


Figure 3.16: Graph of the area comparison between the three algorithms

This is the tradeoff between speed and area. In addition, just removing two carry lookahead adders does not have much of an effect on the total area since there are so many adders in a 32-bit modified non-restoring divider. By using the modified non-restoring division algorithm with the MSC generator, an area of $461 \mu m^2$ has been reduced from modified non-restoring division algorithm with one iteration for 8-bit divider and the former one is almost 10% smaller than the latter one. For 16-bit and 32-bit dividers, the total areas are reduced by 5% and 2.5% respectively.

Table 3.6: Total power consumption for each division algorithm

Power	Standard NRD (a)	Modified NRD with CLA (b)	Modified NRD with MSC (c)	Diff. (a-b)	Diff. (a-c)
8 bit	474.9 µW	955.7 µW	872.9 µW	480.8 µW	398 µW
16 bit	1680.3 µW	3410 µW	3271.6 µW	1729.7 µW	1591.3 µW
32 bit	5957.1 µW	12608.5 µW	12408.2 µW	6651.4 µW	6451.1 µW

Table 3.6 shows the total power consumption including the dynamic power and cell leakage power. For the modified non-restoring division algorithms, doubling the number of the adders allows to reduce the delay, increasing the area size and the power consumption is critical. For the non-restoring division logic with CLAs, the power consumption is twice as much as the standard non-restoring division. In Figure 3.17, the shape of the graph is very similar to that of Figure 3.15 and it is concluded the consumed power is roughly proportional to the area. Therefore, it is unavoidable that the modified non-restoring division algorithm with large area consumes more power than the standard non-restoring division algorithm.

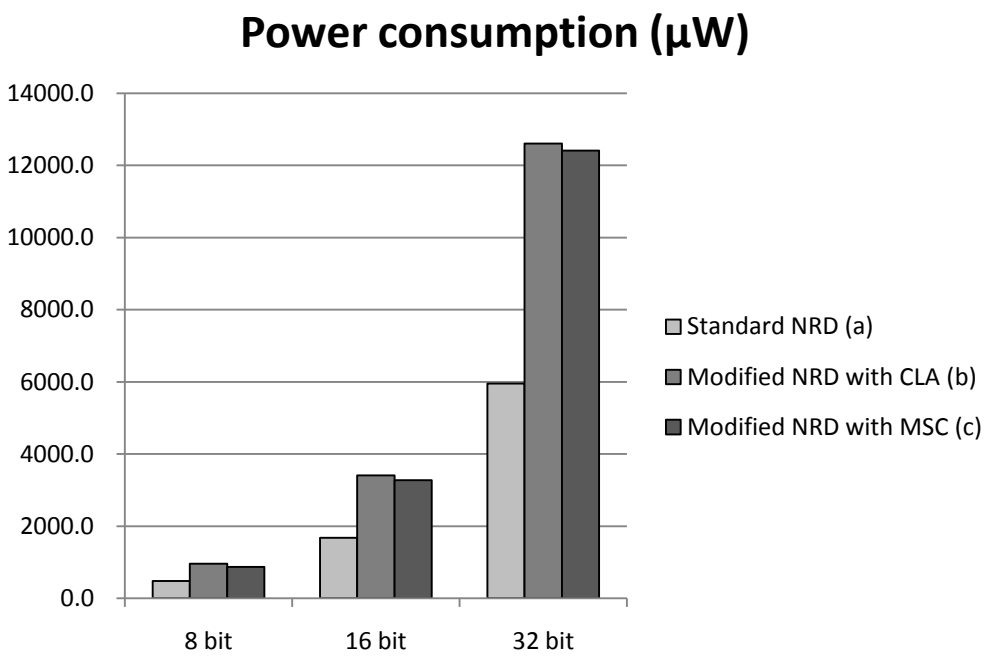


Figure 3.17: Graph of the power consumption for each division algorithm

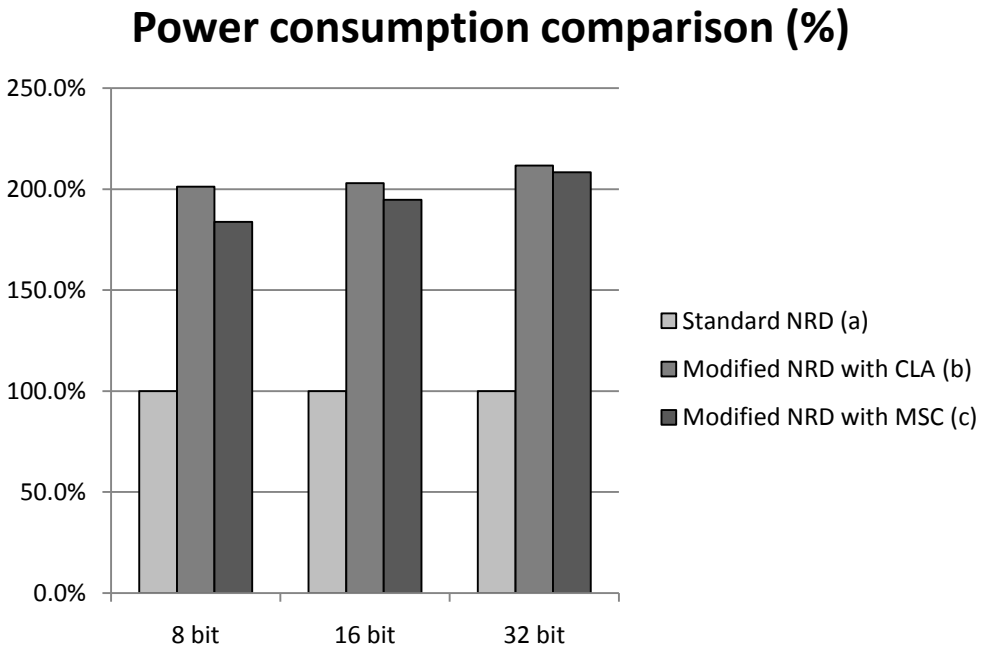


Figure 3.18: Graph of the power consumption comparison between the three algorithms

3.5 SUMMARY

A modified algorithm for non-restoring division has been presented. The modified algorithm increases its speed and enhances its precision as a result of algorithm modifications. It has been verified that the modified non-restoring division reduces the total delay by almost 21% compared to the standard non-restoring division. The reduction increases as the word size increases. The least significant bit of the quotient is correctly generated via the most significant carry generator. The total area is increased by over 78% and the total power consumption is also increased by over 100% as a tradeoff. This modified non-restoring division algorithm has been verified for a 45nm technology using Verilog models and simulations using Synopsys Verilog Compiler Simulator and Design Vision.

Chapter 4

Improved Non-restoring Division Algorithm

4.1 OVERVIEW

The non-restoring division algorithm is the fastest and has least complex of the binary digit recurrence division algorithms [7]. A new method, the modified non-restoring division algorithm to enhance its performance is introduced in Chapter 3. The total delay for the modified non-restoring division algorithm is reduced by almost 21% compared to that of the standard non-restoring division [30]. But, the area and the power consumption are increased by almost 100% since the number of adders used is doubled for the algorithm modification. This problem is very critical and enough to cancel out the speed advantage of the modified non-restoring division algorithm.

If there is a novel idea to reduce the delay of the multiplexer without increasing the number of the adders, it achieves two goals that reduce the total delay without sacrificing the complexity at the same time. So, this research is focused on improving the delay profile for the non-restoring division algorithm without doubling the number of the adders producing the intermediate partial remainders. To accomplish these goals, two new approaches are proposed here. The first approach is the non-restoring division with dual path calculation as shown in Section 4.2. The new algorithm has two different paths. One path is used for finding the sign bit, the most significant bit of the partial remainder, which selects the quotient bit for each iteration and the operand among two possible operands - the denominator and its complement - as an input signal. The other path is used for calculating the partial remainder without the sign bit, which is another operand

entering to the adder in the improved non-restoring division algorithm. In Section 4.3, the modified most significant carry (MSC) generator is introduced. By using it, the 1-bit sign bit is produced before generating the partial remainder without the sign bit from the adder and then it reaches the multiplexer to choose the quotient bit and the operand. It eventually eliminates the total delay of the multiplexer. Finally, the implementation and the simulation results for the improved non-restoring division algorithm are summarized in Section 4.4.

4.2 DUAL PATH CALCULATION

Before explaining the new algorithm, dual path calculation, the typical calculation process to find the partial remainder, P_{n+1}, in the standard non-restoring division algorithm is present as shown in Fig 4.1.

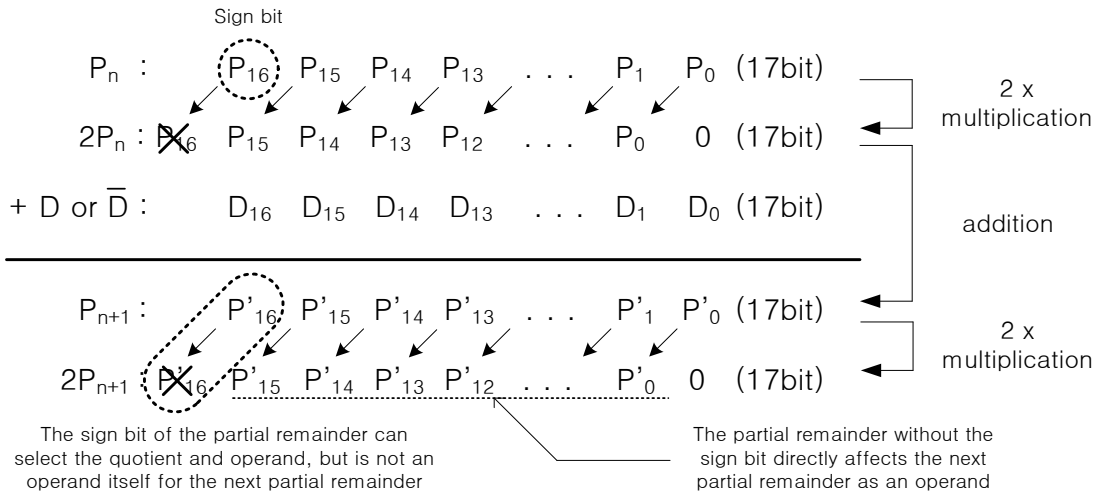


Figure 4.1: Iteration process for the 16-bit standard non-restoring division algorithm

First, the partial remainder for the n-th iteration, P_n, is reaches from the multiplexer to the shifter, and the sign bit, the most significant bit of the partial remainder, is eliminated by a 1-bit left shift process. Instead of it, 0 is inserted at the least significant bit of the partial remainder. That means the sign bit for the partial remainder is no longer necessary for the addition process and it does not affect to the next partial remainder after passing through the multiplexer. The sign bit of the partial remainder can set the quotient digit and the operand among the denominator and its complement, but is not an operand itself for the next partial remainder. So, the new operand for the addition, $2P_n$, does not include the sign bit anymore. The same process is applied to P_{n+1}, the partial remainder for the n+1-th iteration and it happens n times repeatedly.

For both of the operands for addition, P_{15} and D_{16}, at the n-th iteration, they only produce P'_{16}, the sign bit of the next partial remainder. Also it will be removed since it is a leading bit. So, the adder does not have to count on the sign bit of P_{15} and D_{16} if there is another logic for generating it separately. It also reduces the number of bits that the adder handles and eventually reduces the size of the adder. It also makes two separate processes that they can run simultaneously.

Figure 4.2 shows the new iteration process that has two separate paths. One path is used for generating the sign bit of the partial remainder with the modified MSC generator and the other produces the 16-bit wide partial remainder without the sign bit with a 16-bit adder. The dual path algorithm enables two processes to run at the same time. So, another process can be attached after the process with the smaller delay in one path and the process with the larger delay in the other path can hide the delay of the adjacent path. Another difference for the iteration between the standard non-restoring division algorithm and the improved non-restoring division algorithm is the bit width for the adder.

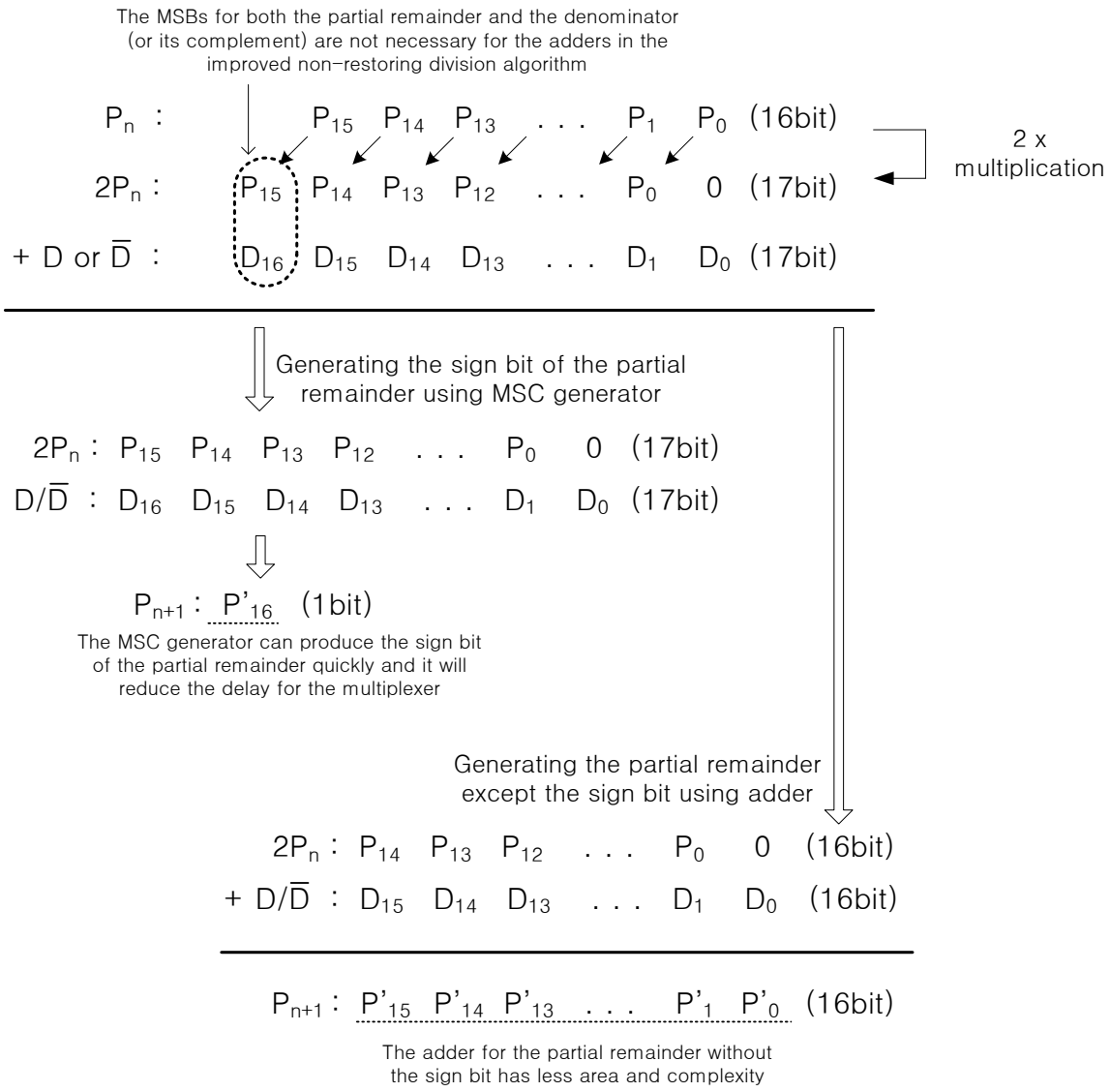


Figure 4.2: Iteration process for the 16-bit improved non-restoring division algorithm

The adder in the standard non-restoring division algorithm receives the n+1 bit wide partial remainder and denominator (or its complement) for its operands since the adder produces the n+1 bit wide partial remainder including the sign bit for the next iteration. The following multiplexer has to wait until the addition is completed and it causes a long delay, but the adder in the improved non-restoring division algorithm only handles n-bit wide numbers and generates the next partial remainder (one bit smaller than that of the standard non-restoring division algorithm.) However, the multiplexer in the improved non-restoring division algorithm does not have to wait for a long delay from the addition process since the modified MSC generator with less delay generates the sign bit separately and simultaneously.

The detailed flow chart for the improved non-restoring division algorithm is presented in Figure 4.3. For each iteration, generating the sign bit and selecting the quotient and the operand based on the sign bit is performed by the multiplexer following the MSC generator and it makes one path marked with the dotted line in Figure 4.3. The other path is a series of addition process including the bit width reduction and multiplication by two drawn with a solid line. The two separate processes run simultaneously. The adder can hide the total delay from the multiplexer and the modified MSC generator since the delay from the adder is longer than that of a series of two logic blocks. For the 16-bit improved non-restoring division algorithm, the delay of the 16-bit adder is 13 unit gate delays while the delay from the 17-bit multiplexer to the 17-bit modified MSC generator is 12 unit gate delays. So, the delay from the adder sets the total delay for each iteration. For the 8-bit improved non-restoring division algorithm, the adder delay is 11 unit gate delays and it is identical to the combined delay of the multiplexer and the MSC generator. In this case, the two processes run through each path

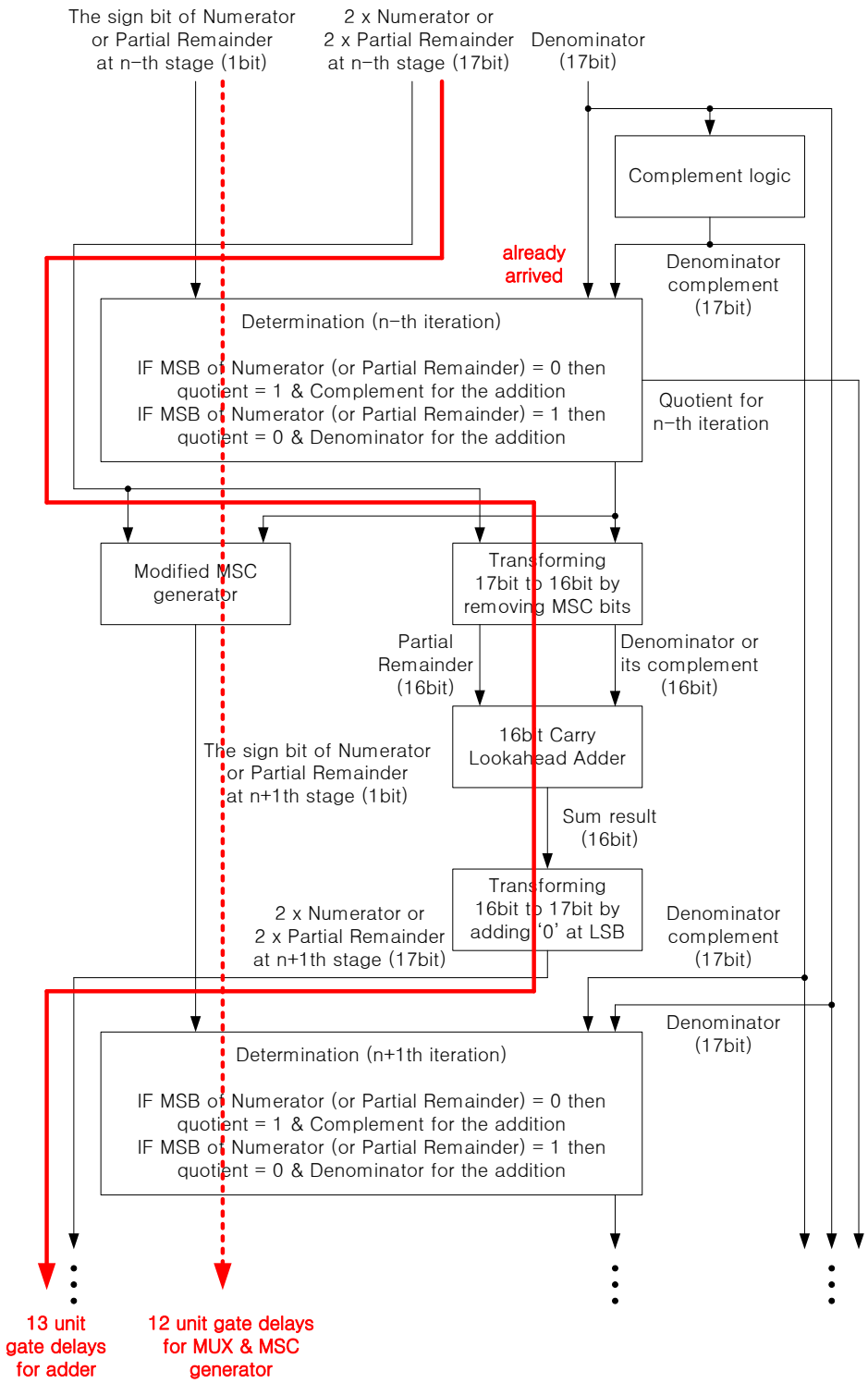


Figure 4.3: The flowchart for the 16-bit improved non-restoring division algorithm

with the same delay, so the total delay for each iteration is 11 unit gate delays. For the 32-bit division algorithm, the adder with 17 unit gate delays has a slower delay profile than the combined logic blocks with 13 unit gate delays provided the denominator (or its complement) arrives at both the adder block and the MSC generator at the same time. Therefore the delay from the 32-bit adder can be assumed to be the total delay. To sum up, the delay from the adder is the total delay for each iteration on all of the 8, 16 and 32-bit improved non-restoring division algorithms.

The advantage of the improved non-restoring division algorithm is very clear by comparing with other non-restoring division algorithms. Figure 4.4 shows the process flow for the standard non-restoring division algorithm as shown in Chapter 2 [10]. Once the partial remainder arrives the multiplexer, the multiplexer selects the quotient as well as the second operand other than the partial remainder. The second operand, the denominator or its complement, can enter to the adder after the multiplexer finishes to select them. It means the adder has to wait until the multiplexer has done its process. Therefore, it is evident that the series of processes in the standard non-restoring division algorithm is fully sequential and the total delay for each iteration is the sum of the delays for both the multiplexer and the adder.

The modified non-restoring division algorithm is shown in Figure 4.5. It has partly simultaneous processes that overlap each other partially. The partial remainder enters to the multiplexer first and the most significant bit of the partial remainder acts as a select signal. The delay of the multiplexer is 3 unit gate delays if the select signal enters as an input signal. Then the multiplexer waits for the two data signals from the adders and the delay of the multiplexer is two with the data signals since they only pass through two gates in the multiplexer as is shown in Figure 3.2. Therefore, the total delay for each iteration is one less than the sum of the delays for both the multiplexer and the adder.

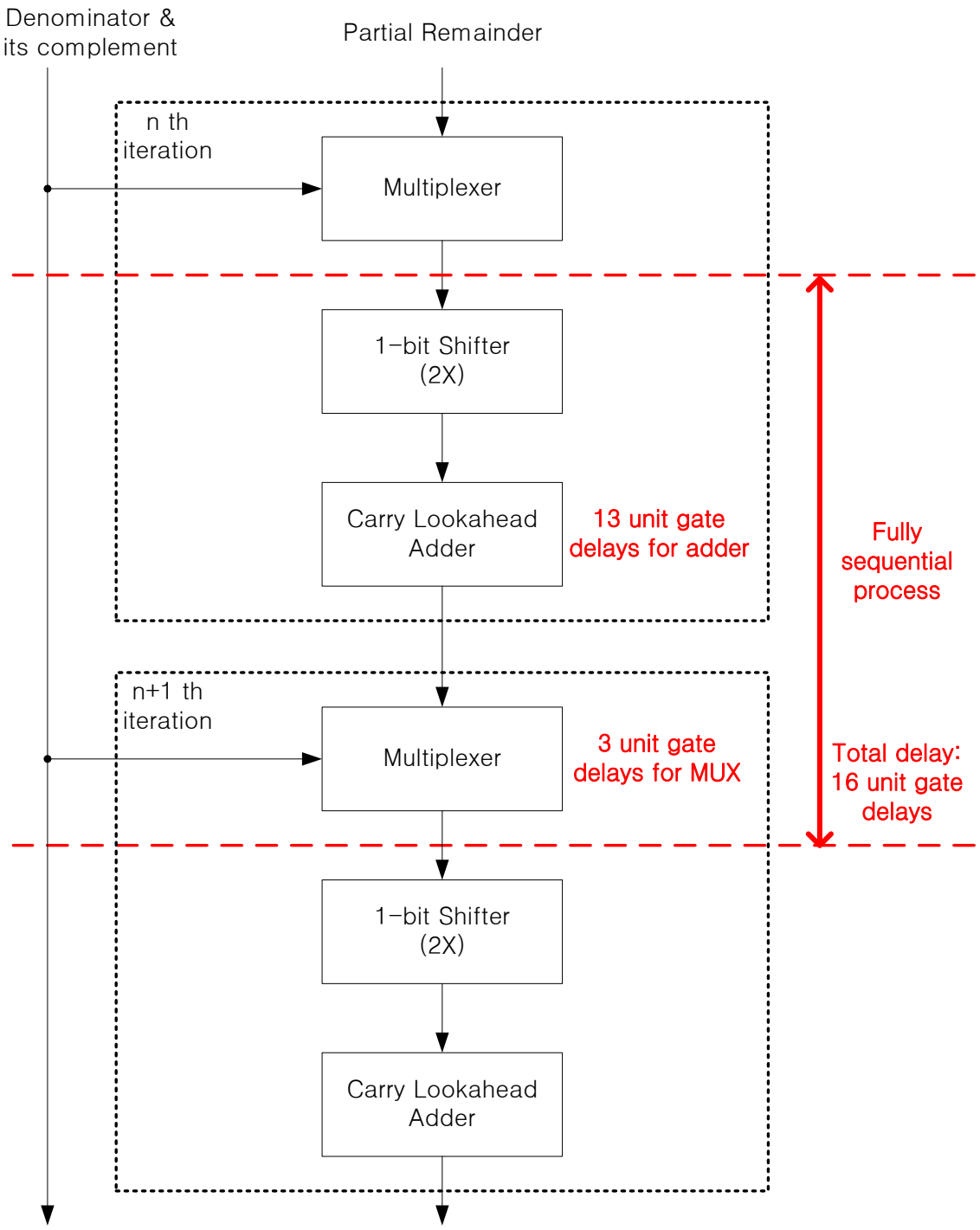


Figure 4.4: Process flow for the 16-bit standard non-restoring division algorithm

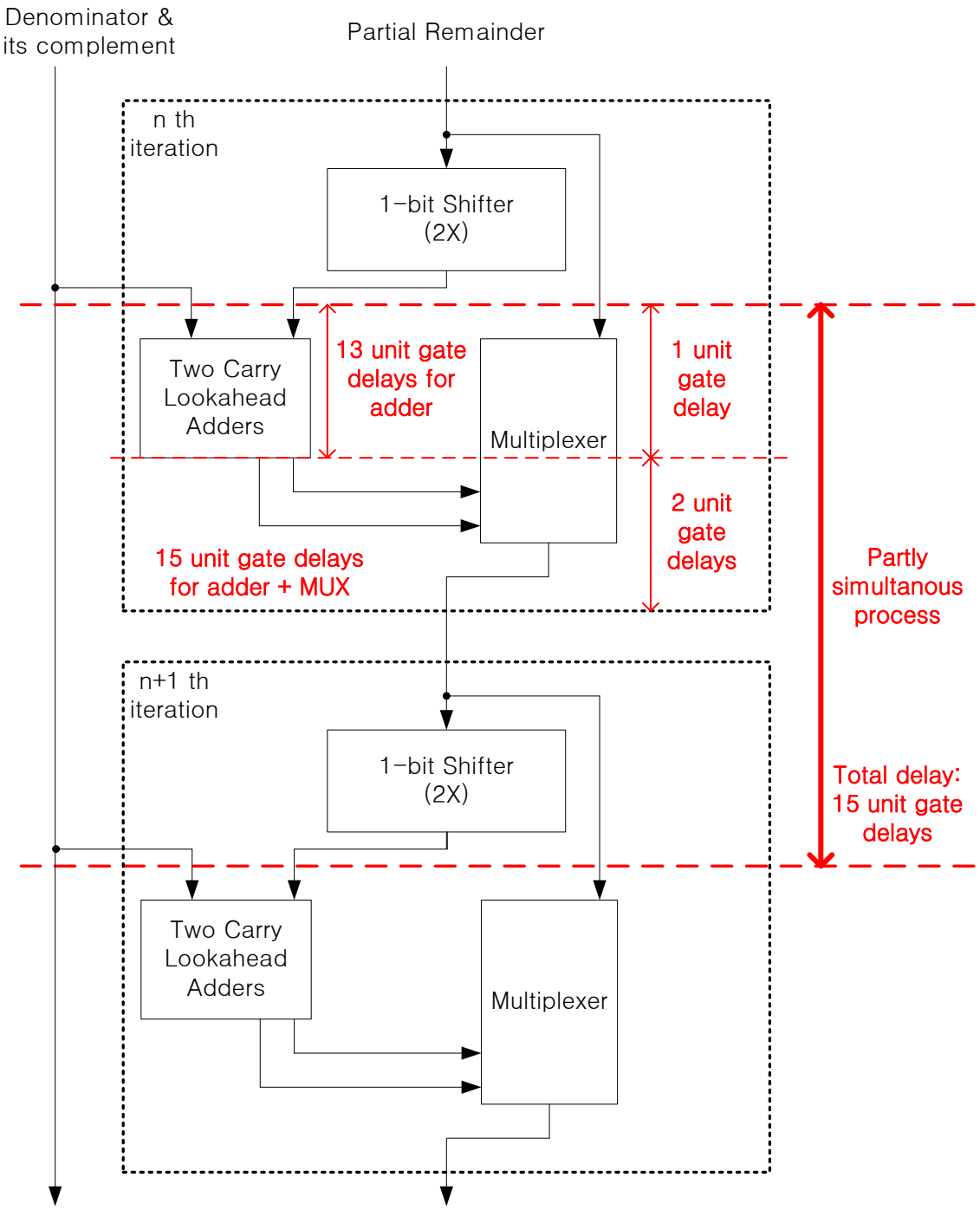


Figure 4.5: Process flow for the 16-bit modified non-restoring division algorithm

Figure 4.6 shows the process flow for the 16-bit improved non-restoring division algorithm. The modified MSC generator and the adder start to generate the partial remainder after receiving the operand from the multiplexer. The sign bit of the partial remainder is produced first from the modified MSC generator since it only takes 9 unit gate delays compared to 13 unit delays for the adder. The sign bit then enters to the multiplexer to select the operand as a select signal and the multiplexer then generates the operand and pass it to both the adder and the modified MSC generator for the next iteration. The partial remainder from the adder at current stage arrives after completing a series of process on the other path. So, the improved non-restoring division algorithm runs two processes at the same time using the dual path and the total delay is reduced to 13 unit gate delays per iteration. It is assumed either a 2-input AND or OR gate or an inverter has one unit gate delay.

4.3 MODIFIED MOST SIGNIFICANT CARRY GENERATOR

Finding the most significant carry of the result of an addition is necessary in some applications including the sign detection for the division algorithm since the most significant carry of the partial remainder can set the quotient and select the operand for the following addition process for the next partial remainder as shown in Figure 3.8. So, for the improved non-restoring division algorithm, it is important to find the most significant carry before the result of the addition produced.

The most significant carry is generally computed by CLA-based implementation, but the delay can be too high for some applications and it is 20% more than the critical path delay compared to most significant carry detection by reverse carry method [12, 17, 18, 19].

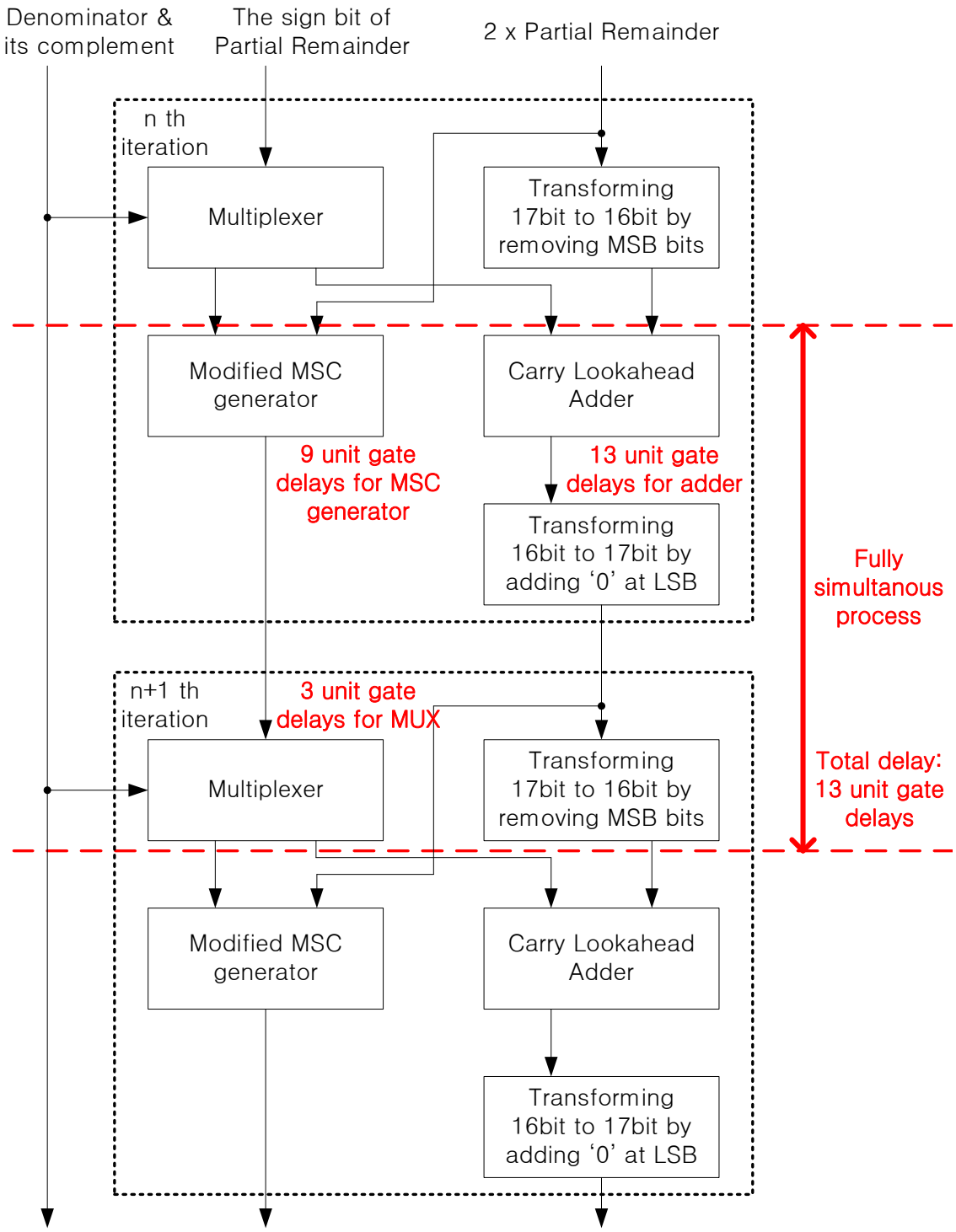


Figure 4.6: Process flow for the 16-bit improved non-restoring division algorithm

So, the MSC detection by the reverse carry method is used for the improved non-restoring division algorithm, the same method for the modified non-restoring division algorithm in Chapter 3. This algorithm has three basic steps. There are determining the beginning of the most significant carry propagation chain, determining the value of the carry and collecting the carry. Before explaining the modified MSC generator in detail, the basics of the MSC generator, already mentioned in Chapter 3, are reviewed and presented briefly.

Based on the non-restoring algorithm, the positive current remainder is always paired with a negative denominator, the ones complement of the denominator, to calculate the next partial remainder as already shown in Figure 3.8. On the other hand, the negative current remainder is always paired with a positive denominator to calculate the next partial remainder as shown in Figure 3.8. In other words, the most significant bit of the denominator, +D, is always zero if the most significant bit of the doubled partial remainder, P_i, is one. Similarly, the most significant bit of the two's complement of the denominator, -D, is always one if the most significant bit of the doubled partial remainder, P_i, is zero. Since the extended sign bit of the next partial remainder, P_{i+1}, decides whether it is greater than zero or less than zero, the carry resulting from the addition at n+1-th bit is one, then the extended sign bit of the next partial remainder, P_{i+1}, is set to zero and the final quotient is generated as one. If the carry is not propagated to the extended sign bit, the partial remainder is assumed to be less than zero and the last quotient is zero.

The algorithm for the MSC generator modified for the non-restoring division is shown in Figure 4.7. The intermediate results k_i and g_i are generated by OR and AND logic operations respectively. To find the beginning position of the most-significant carry chain, h_i is generated by an AND operation from the most significant bit to each of the

remaining bits in order. Then, the value of the carry determined by d_i performing an AND operation with the previously generated g_{i+1} and h_i. Finally, collect the MSC if there is at least one d_i =1 and it can be generated by OR operations from the most significant bit to each of the remaining bits in order.

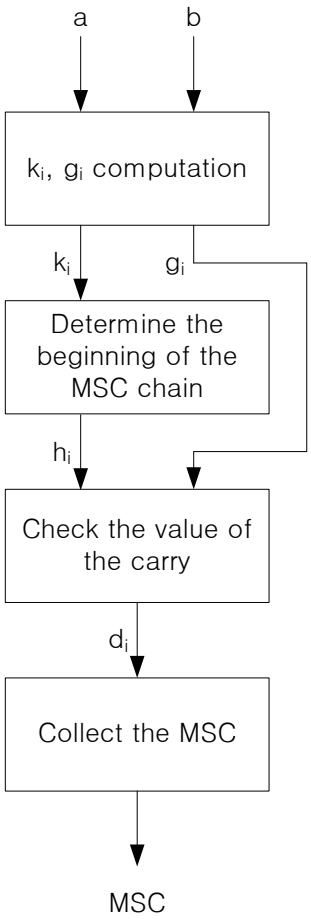


Figure 4.7: Steps in calculation of the MSC by reverse carry

The doubled partial remainder $2P_i$ and the denominator +D (or its complement -D) are used as an operands for producing the next partial remainder $2P_{i+1}$. The,

subtraction process, $2P_i$ - D, is performed if P_i is positive. Otherwise, the addition process, $2P_i$ + D, is required. The subtraction can be replaced by the addition with the 2's complement of the one of two operands, and adding the adjustment digit to the 1's complement of an operand is a required process for the 2's complement. For the standard non-restoring division algorithm, the adder with a carry input successfully performed this required process for the 2's complement by treating the adjustment digit as a carry as shown in Figure 4.8.

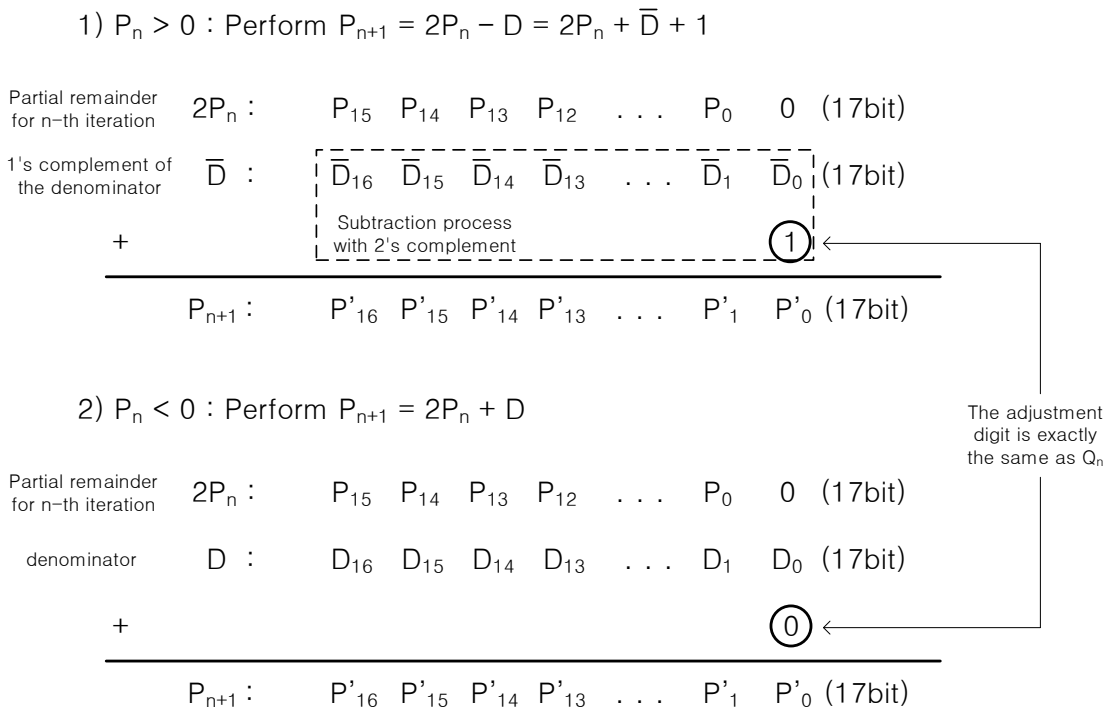


Figure 4.8: The addition process for the standard non-restoring division algorithm

The same operands are also required for the MSC generation process, but there is an issue that arise regarding the adjustment digit. Since the MSC generator can only deal

with two operands, the doubled partial remainder $2P_i$ and the denominator +D (or its complement -D), it cannot handle the adjustment digit properly unlike the adder with three inputs. Therefore, one more addition is necessary for converting the 1's complement number to the 2's complement number by adding one if the partial remainder is positive and it will increase the delay and the area considerably. So, the alternative method for converting a 1's complement to a 2's complement without an addition is required for the MSC generator to find the correct sign bit.

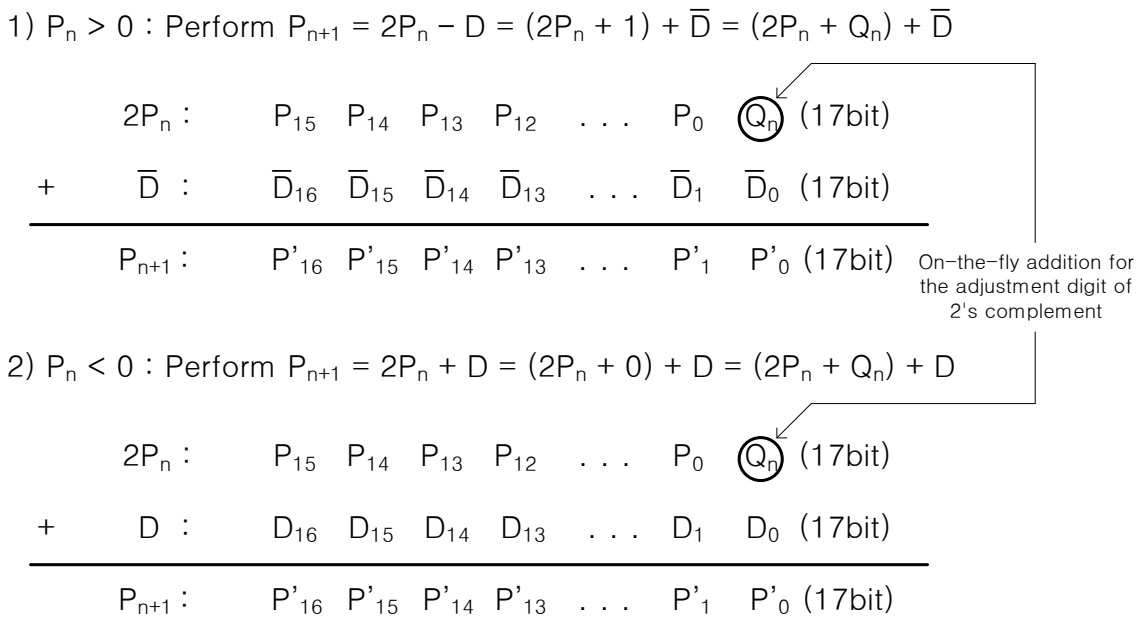


Figure 4.9: The modified MSC generation process for the improved non-restoring division algorithm

Figure 4.9 shows the new method to deal with the adjustment digit for the 2's complement number is presented. One of the operands for the non-restoring division

algorithm is the doubled partial remainder $2P_i$ from the 1-bit left shifter and its least significant bit is always zero. So, the addition of the doubled partial remainder and the adjustment digit for the complement of the denominator can be replaced by simply putting the adjustment digit to the least significant bit of $2P_i$. The current quotient digit Q_n, is used for the adjustment digit because the current quotient digit is set to one if the doubled partial remainder is positive. The adjustment digit is one if the complement of the denominator is used for the one of the operands for the addition process, and this situation happens unless the doubled partial remainder is less than zero. Therefore, the current quotient digit Q_n, can be used as the adjustment digit. This proposed process for the MSC generation, on-the-fly addition, deals with the 2's complement number as its operand correctly without an unnecessary addition.

A new algorithm for the MSC generator with on-the-fly addition functionality has been presented. The modified MSC generator generates the correct sign bit without sacrificing the delay profile and increasing the area.

4.4 VERIFICATION AND SIMULATION RESULTS

In this section, both analysis and simulation are performed for verification purposes. Nine different division circuits are implemented and analyzed in order to confirm the proposed algorithm is better than the previous ones with respect to the estimated delays and complexities. Then, the simulations are performed using the Verilog Hardware Description Language (Verilog HDL). The FreePDK45nm version 1.0 is also used as a library file for the delay, the area and the power consumption.

Table 4.1: The estimated unit gate delays and the complexities for 3 different 8-bit division algorithms

	8bit standard NRD			8bit modified NRD			8bit improved NRD		
	Components	Delay	Complexity	Components	Delay	Complexity	Components	Delay	Complexity
1st stage	9bit inverter	1Δ	9	9bit inverter	1Δ	9	9bit inverter	1Δ	9
	9bit MUX	2Δ	28	9bit CLA	11Δ	124	9bit MSC generator		43
	9bit CLA	11Δ	124				8bit CLA	11Δ	113
Inter-mediate stages (2nd to 7th stage)	9bit MUX	18Δ	168	2 9bit CLAs	66Δ	1488	9bit MUX		168
	9bit CLA	66Δ	744	9bit MUX	12Δ	168	9bit MSC generator		258
							8bit CLA	66Δ	678
Final stage (8th stage)	9bit MUX	3Δ	28	9bit MUX	3Δ	28	9bit MUX		28
	9bit CLA	11Δ	124	9bit MSC generator	8Δ	43	9bit MSC generator	8Δ	43
	1bit inverter	1Δ	1						
Total	-	113Δ	1226	-	101Δ	1860	-	86Δ	1340
Percen-tage (%)		100.0%	100.0%		89.4%	151.7%		76.1%	109.3%

Table 4.2: The estimated unit gate delays and the complexities for 3 different 16-bit division algorithms

	16bit standard NRD			16bit modified NRD			16bit improved NRD		
	Components	Delay	Complexity	Components	Delay	Complexity	Components	Delay	Complexity
1st stage	17bit inverter	1Δ	17	17bit inverter	1Δ	17	17bit inverter	1Δ	17
	17bit MUX	2Δ	52	17bit CLA	13Δ	249	17bit MSC generator		85
	17bit CLA	13Δ	249				16bit CLA	13Δ	238
Inter-mediate stages (2nd to 15th stage)	17bit MUX	42Δ	728	2 17bit CLAs	182Δ	6972	17bit MUX		728
	17bit CLA	182Δ	3486	17bit MUX	28Δ	728	17bit MSC generator		1190
							16bit CLA	182Δ	3332
Final stage (16th stage)	17bit MUX	3Δ	52	17bit MUX	3Δ	52	17bit MUX		52
	17bit CLA	13Δ	249	17bit MSC generator	9Δ	85	17bit MSC generator	9Δ	85
	1bit inverter	1Δ	1						
Total	-	257Δ	4834	-	236Δ	8103	-	205Δ	5727
Percen-tage (%)		100.0%	100.0%		91.8%	167.6%		79.8%	118.5%

Table 4.3: The estimated unit gate delays and the complexities for 3 different 32-bit division algorithms

	32bit standard NRD			32bit modified NRD			32bit improved NRD		
	Components	Delay	Complexity	Components	Delay	Complexity	Components	Delay	Complexity
1st stage	33bit inverter	1Δ	33	33bit inverter	1Δ	33	33bit inverter	1Δ	33
	33bit MUX	2Δ	100	33bit CLA	17Δ	492	33bit MSC generator		172
	33bit CLA	17Δ	492				32bit CLA	17Δ	481
Inter-mediate stages (2nd to 31st stage)	33bit MUX	90Δ	3000	2 33bit CLAs	510Δ	29520	33bit MUX		3000
	33bit CLA	510Δ	14760	33bit MUX	60Δ	3000	33bit MSC generator		5160
							32bit CLA	510Δ	14430
Final stage (32nd stage)	33bit MUX	3Δ	100	33bit MUX	3Δ	100	33bit MUX		100
	33bit CLA	17Δ	492	33bit MSC generator	10Δ	172	33bit MSC generator	10Δ	172
	1bit inverter	1Δ	1						
Total	-	641Δ	18978	-	601Δ	33317	-	538Δ	23548
Percen-tage (%)		100.0%	100.0%		93.8%	175.6%		83.9%	124.1%

4.4.1 Algorithm Analysis

The estimated delays and complexities for 9 different non-restoring dividers are presented in Table 4.1, 4.2 and 4.3. The division algorithms can be conveniently divided into three sections. The first stages that occurr in the first iteration have a couple of differences compared to the intermediate stages. They have the complement logic that consists of n+1 inverters with n+1 bit width. For both the modified and the improved non-restoring division algorithms, the multiplexer for selecting the first quotient q_0 is removed since it is a leading bit and eventually eliminated through the on-the-fly digit conversion process. For the intermediate stages, each division algorithm produces an n-2 quotients using its own process. Whereas a multiplexer and an adder are used for the standard non-restoring division algorithm, the modified non-restoring division algorithm uses a multiplexer and twice as many adders as the standard non-restoring division algorithm. The improved non-restoring division algorithm uses both the MSC generator and the adder to find the partial remainder at the same time. Therefore, the standard non-restoring division algorithm has less complexity among the 3 division algorithms, but the improved non-restoring division algorithms have a lower unit delay due to a fully simultaneous process. For the final stage, a multiplexer, an adder and an inverter are used for correcting the possible quotient error of the least significant bit for the standard non-restoring division algorithm and it has most complex and slowest final stage of the three division algorithms. The other two division algorithms use a multiplexer, an MSC generator to reduce the delay and the complexity. The improved non-restoring division algorithm minimizes the delay for the final stage since the adder for previous iteration has a long delay and it hides the delay of the multiplexer as a result. Please note that either a 2-input AND or OR gate or an inverter is counted as 1 gate for the calculating the total

complexity. From the delay perspective, either a 2-input AND or OR gate or an inverter has 1 unit gate delay.

First, 3 different 8-bit division algorithms are analyzed as shown in Table 4.1. The improved non-restoring division algorithm is the fastest and it has 23.9% less unit gate delay compared to the standard non-restoring division algorithm. The modified non-restoring division algorithm, the largest division algorithm of three, sacrifices complexity by doubling the number of the adders for an increase its speed. As a result, its area is increased by 51.7%, but the area for the improved non-restoring division algorithm is only increased by 9.3% although the improved algorithm is faster than the modified one by 13.3%. To sum up, the improved non-restoring division algorithm reduces its unit gate delay by 23.9% and it increases its complexity by only 9.3%. So, it has better overall performance than the modified non-restoring division algorithm.

For 16 and 32-bit division algorithm comparison chart as shown in Table 4.2 and 4.3, the improved non-restoring division algorithm has better overall performance with respect to both the delay and the complexity than the modified non-restoring division algorithm. The total unit gate delays are reduced by 20.2% and 16.1% for 16 and 32-bit improved non-restoring division, respectively while the modified non-restoring division algorithm reduces its delays by 8.2% and 6.2% compared to the standard non-restoring division. The advantage is even greater if the complexity for each division algorithms is analyzed. The complexities for the modified non-restoring division algorithm is increased by 67.6% and 75.6% for 16 and 32-bit improved non-restoring division respectively and they are almost twice complexity compared to the standard non-restoring division algorithm, but the 16 and 32-bit improved non-restoring division algorithms raise their complexities by only 18.5% and 24.1% respectively and these numbers are a third of that of the modified non-restoring division. The analysis shows the improved non-restoring

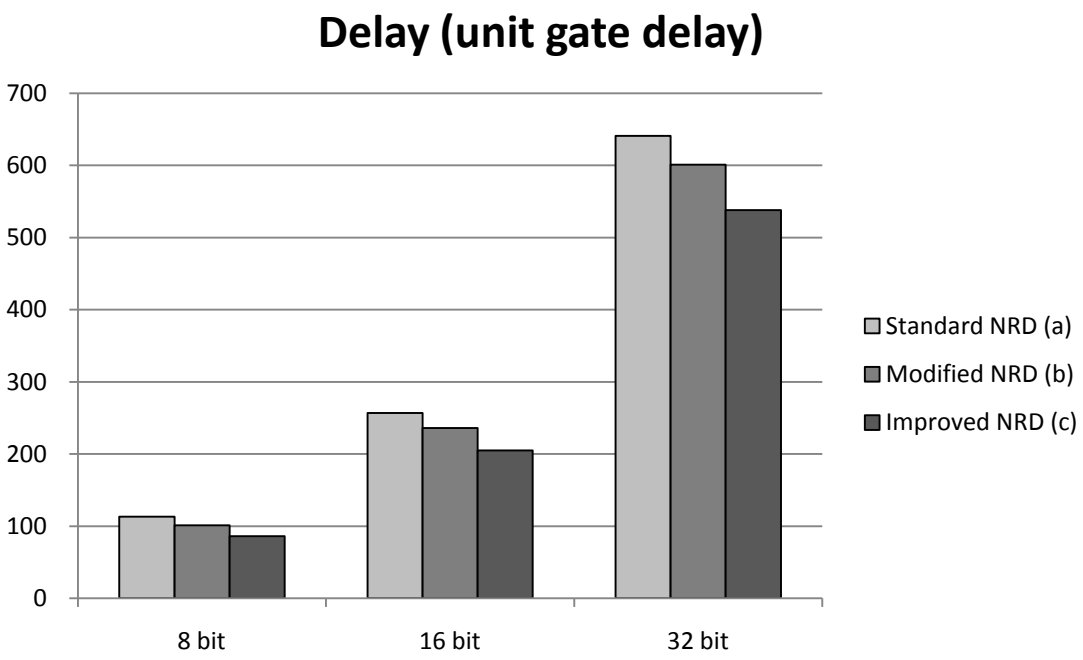


Figure 4.10: Graph of the estimated delays for each division algorithm

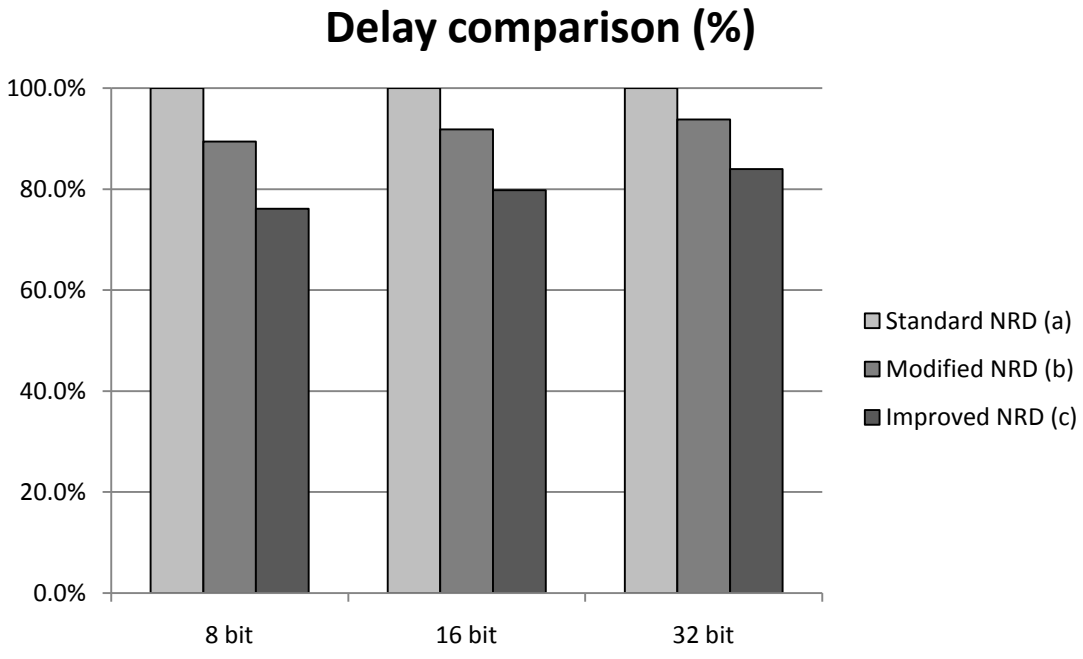


Figure 4.11: Graph of the estimated delay comparison between the three algorithms

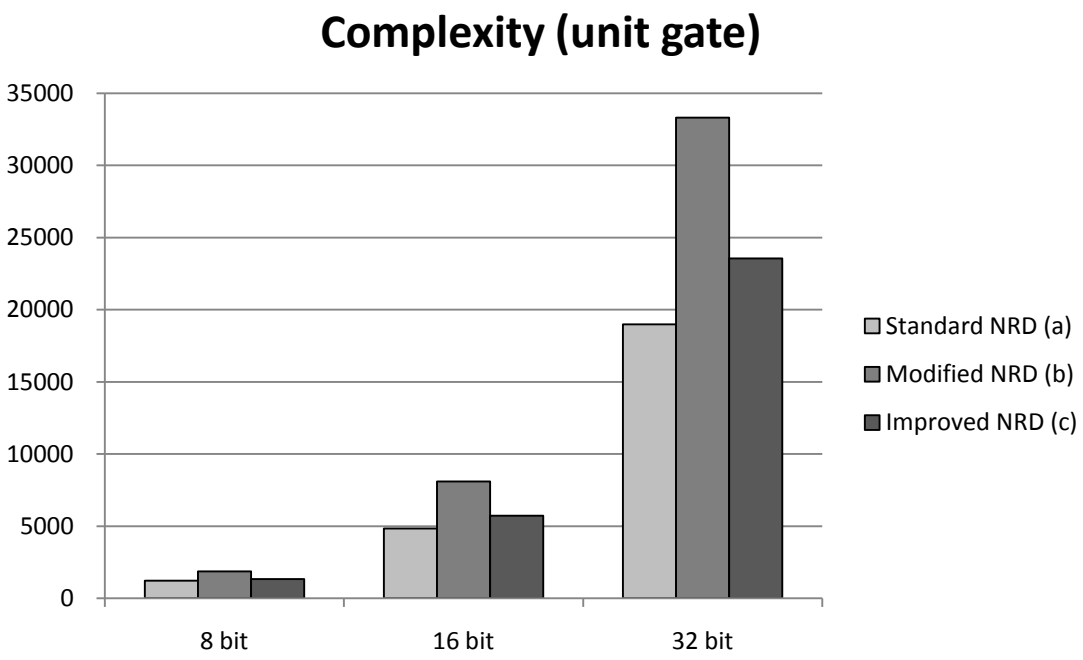


Figure 4.12: Graph of the complexities for each division algorithm

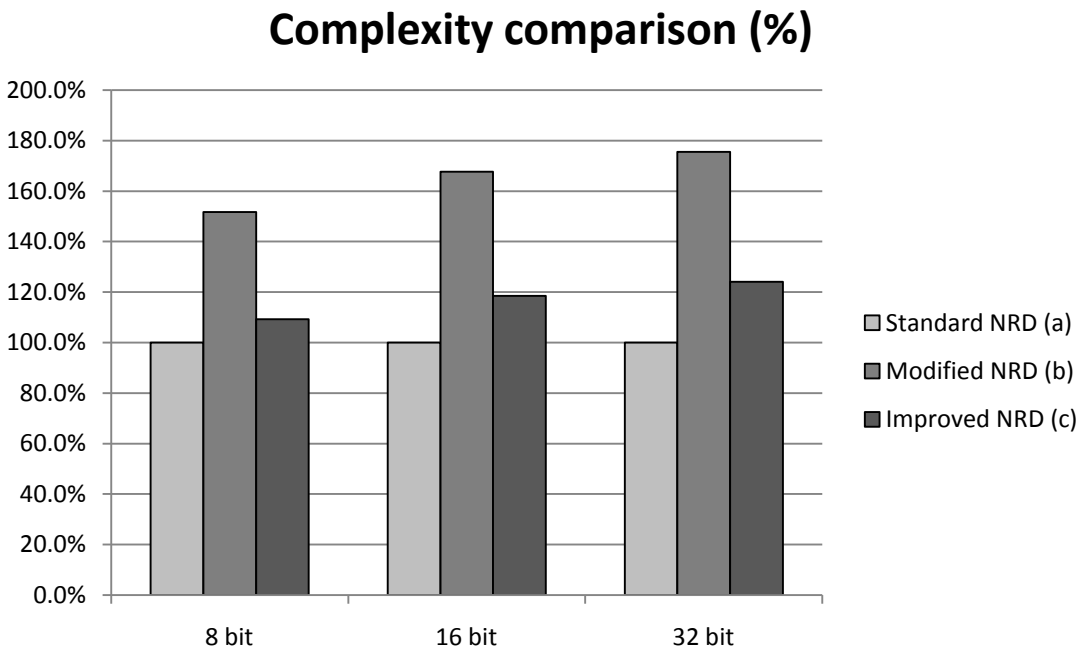


Figure 4.13: Graph of the complexity comparison between the three algorithms

division algorithm reduces the delay by 23.9% minimizing the increase of its complexity as the tradeoff. This analysis is checked and compared to the simulation result to find the non-restoring division algorithm with the best performance.

4.4.2 Simulation Results

Nine different division circuits are implemented using the Verilog Hardware Description Language (Verilog HDL). The FreePDK45nm version 1.0 is used as a library file for the delay, the area and the power consumption. The Synopsys Verilog Compiler Simulator (VCS) program is used for finding the dynamic delays with various Verilog HDL files for the dividers. Design Vision is also used for finding the area and the total power consumption including the total dynamic power and the cell leakage power. A hundred random vectors are generated as both the numerators and the denominators for each simulation and the delays in Table 4.4 are the average values of a hundred simulations.

Table 4.4: Average delays for each division algorithm

delay	Standard NRD (a)	Modified NRD (b)	Improved NRD (c)	Diff. (a-b)	Diff. (a-c)
8 bit	21.04 ns	16.66 ns	16.15 ns	4.38 ns	4.89 ns
16 bit	51.17 ns	40.30 ns	39.25 ns	10.87 ns	11.92 ns
32 bit	112.22 ns	88.29 ns	85.71 ns	23.93 ns	26.51 ns

Table 4.4 shows that the improved non-restoring division algorithm is the fastest compared to both the standard and the modified non-restoring division algorithm. For the 8-bit comparison, the improved non-restoring division algorithm has a delay of 16.15 nanoseconds, which is 23.2% less than the standard and this result is almost identical to

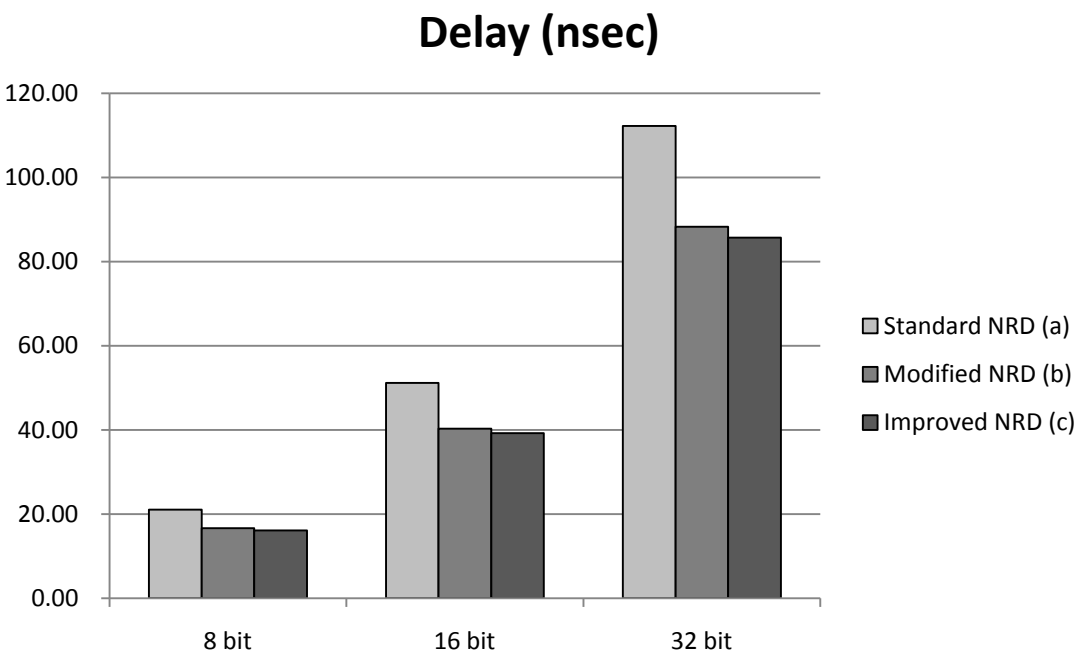


Figure 4.14: Graph of the delays for each division algorithm

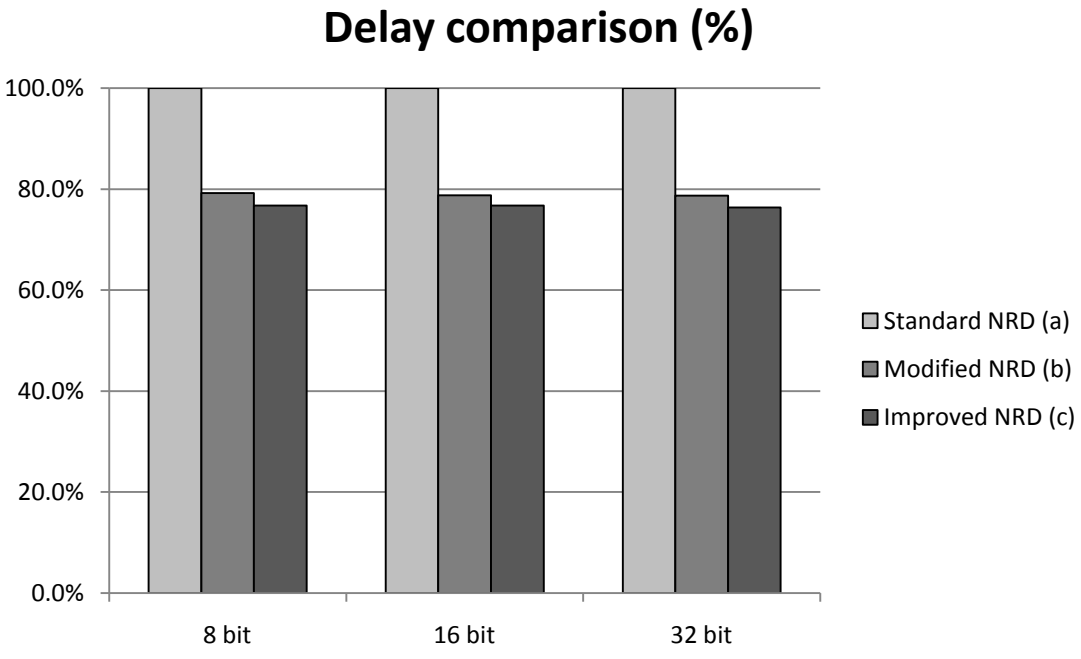


Figure 4.15: Graph of the delay comparison between the three algorithms

the result from the analysis done in Section 4.4.1 that predicts the improved division algorithm has 23.9% less delay than the standard. For 16-bit and 32-bit division algorithms, the delays are reduced by 23.2% and 23.6% respectively as shown in Figure 4.15. The reduction in the total delay becomes slightly larger as the number of bits is increased contrary to the analysis result. The analysis in Section 4.4.1 predicts the unit gate delays are reduced by 20.2% and 16.1% respectively for the 16-bit and 32-bit division algorithms and their reduction will become smaller as the number of bits is increased. The inaccurate approximation may lead to the discrepancy between the simulation and analysis results since only the logic circuits are counted for the analysis.

Table 4.5: Areas for each division algorithm

Area	Standard NRD (a)	Modified NRD (b)	Improved NRD (c)	Diff. (a-b)	Diff. (a-c)
8 bit	$2590\,\mu m^2$	$3999\,\mu m^2$	$2715\,\mu m^2$	$1409\ \mu m^2$	$125\ \mu m^2$
16 bit	$10290\,\mu m^2$	$17556\,\mu m^2$	$11895\,\mu m^2$	$7266\ \mu m^2$	$1605\ \mu m^2$
32 bit	$40806\,\mu m^2$	$72945\,\mu m^2$	$50132\,\mu m^2$	$32139\ \mu m^2$	$9326\ \mu m^2$

For the modified non-restoring division algorithm discussed in Chapter 3, the major disadvantage is the larger size and it has almost twice as many adders as the standard non-restoring division algorithm. This disadvantage is enough to cancel out the advantage of a delay reduction of up to almost 22%. To mitigate it, the MSC generator is introduced and the smaller adder with the MSC generator can replaces the two larger adders for the modified non-restoring division algorithm. In Table 4.5, the 8-bit improved non-restoring division algorithm has an area of $2715\mu m^2$ whereas the standard division

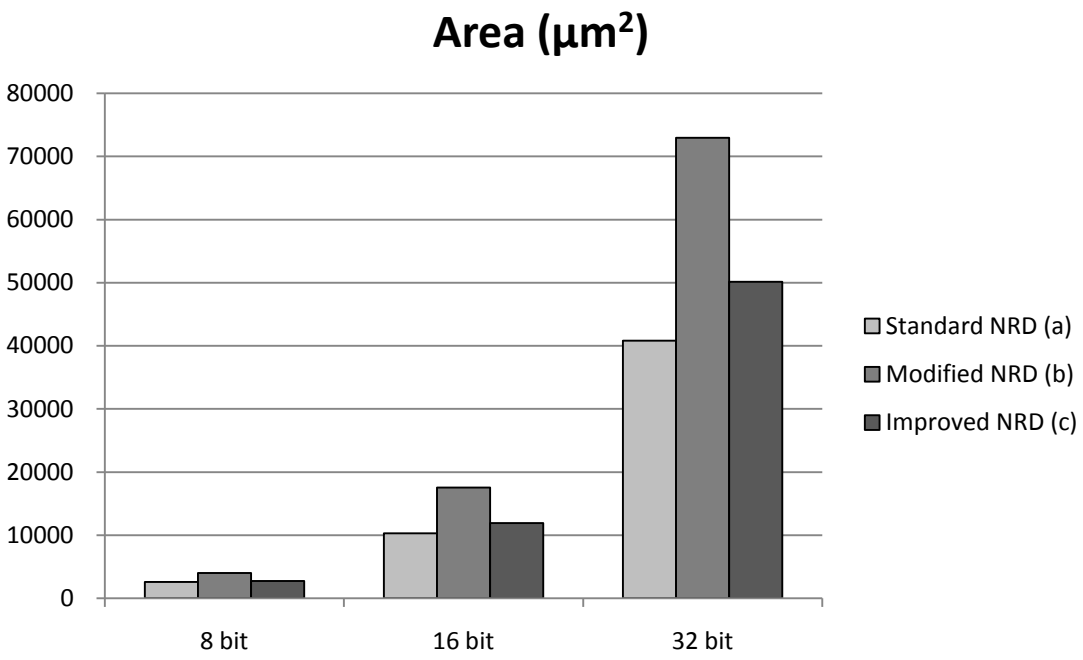


Figure 4.16: Graph of the areas for each division algorithm

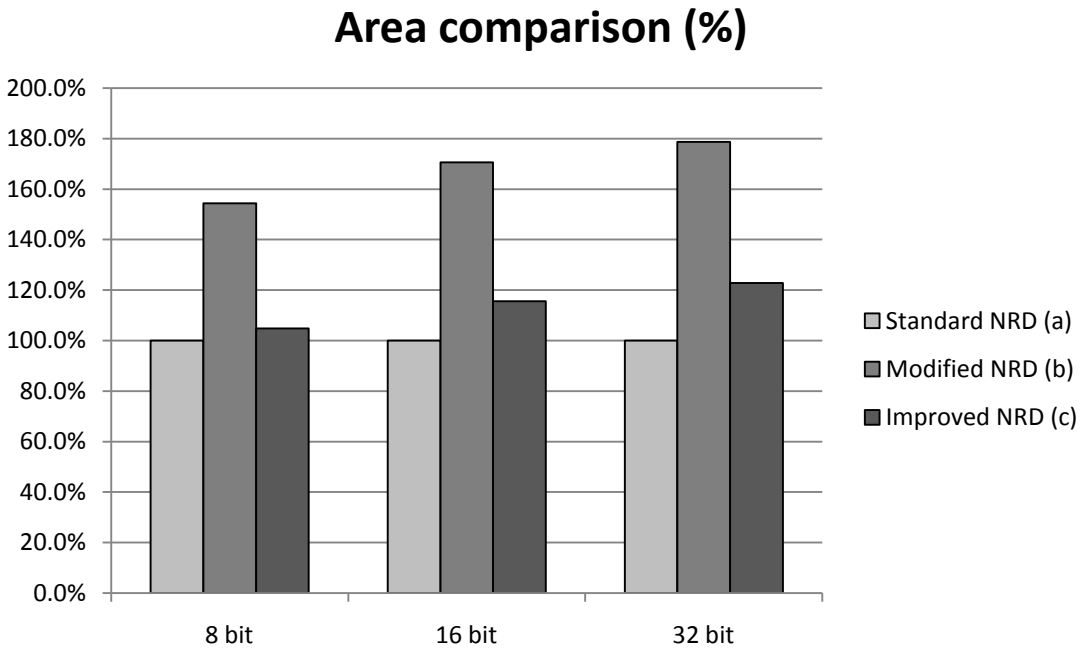


Figure 4.17: Graph of the area comparison between the three algorithms

algorithm has an area of 2590μm^2 which is only a 4.8% increase as shown in Figure 4.17. For 16-bit and 32-bit dividers, the areas are increased by 15.6% and 22.9%, respectively. From analysis of the complexities for each division algorithm, it turns out the complexities are increased by 9.3%, 18.5% and 24.1%, respectively for the 8, 16 and 32-bit improved non-restoring division algorithms when compared to the standard non-restoring division algorithms. So, both the simulated and the analyzed results are very similar with respect to the areas. It is concluded that the improved algorithm has less area than the modified algorithm and it successfully reduces the size without increasing the delay.

Table 4.6: Total power consumption for each division algorithm

Power	Standard NRD (a)	Modified NRD (b)	Improved NRD (c)	Diff. (a-b)	Diff. (a-c)
8 bit	474.9 μW	872.9 μW	630.9 μW	398 μW	156 μW
16 bit	1680.3 μW	3271.6 μW	2647.4 μW	1591.3 μW	967.1 μW
32 bit	5957.1 μW	12408.2 μW	10649.6 μW	6451.1 μW	4692.5 μW

Table 4.6 shows that the modified non-restoring division algorithm consumes more power than the other division algorithms and it consumes over twice as much power as the standard non-restoring division. While the improved non-restoring algorithm has less power consumption by 27% compared to the modified algorithm, it still consumes 32% more power for the 8-bit divider comparison. The increase in the power consumption increases with the word size and it surpass the area increase. This is a drawback of the improved non-restoring division algorithm and also a tradeoff between

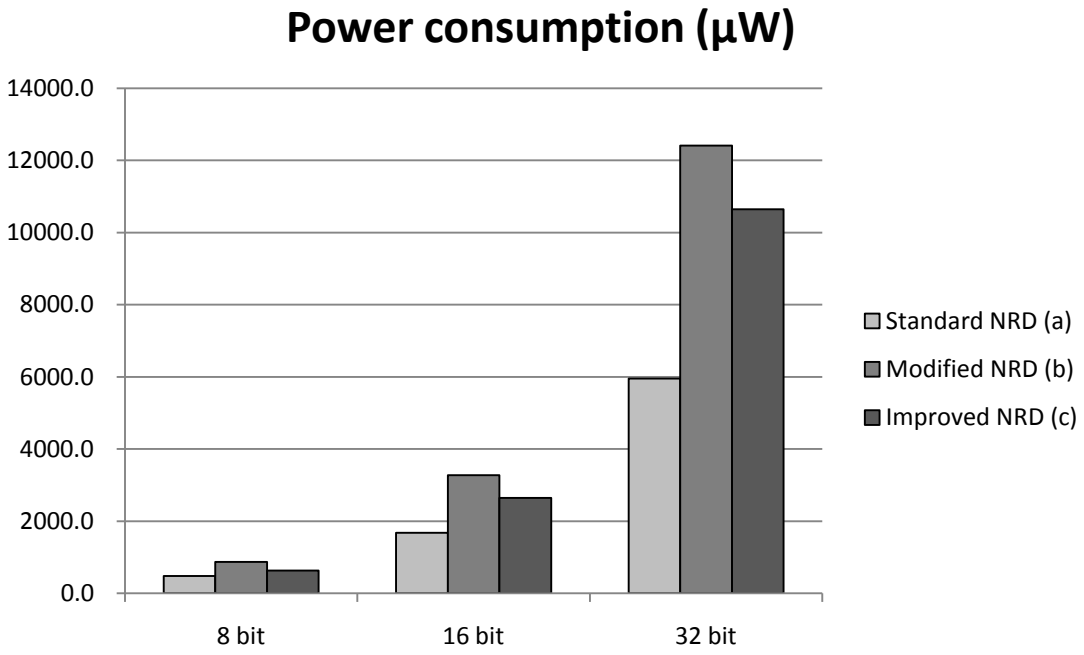


Figure 4.18: Graph of the power consumption for each division algorithm

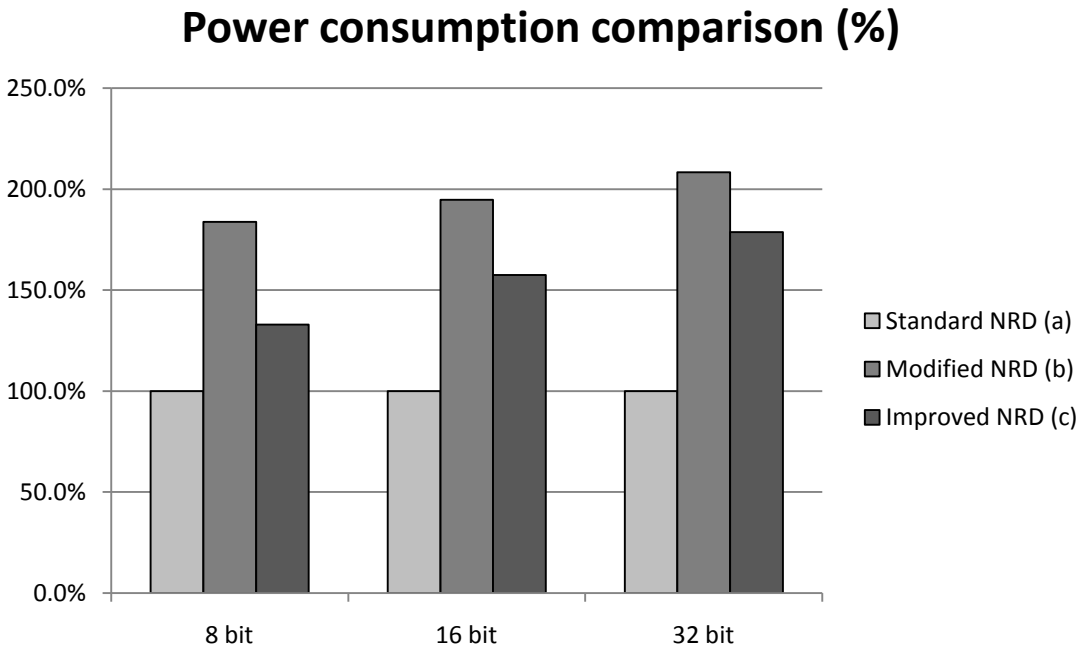


Figure 4.19: Graph of the power consumption comparison between the three algorithms

speed and power consumption. In the future, research focusing on minimizing the power consumption needs to be conducted.

4.5 SUMMARY

A new algorithm for the non-restoring division that performs faster division calculation than the modified division algorithm has been presented. The improved non-restoring division algorithm increases its speed and minimizes the area and the power consumption compared to the modified non-restoring division algorithm. The main contribution of this proposed method is the dual path structure that enables the divider to perform two different processes simultaneously and that is also made possible by the modified MSC generator. It has been verified that the improved non-restoring division reduces the total delay by up to 24% compared to the standard non-restoring division. The improved non-restoring division algorithm has almost 32% less area and up to 27% less power consumption than the modified non-restoring division although it still has a larger area and higher power consumption than the standard. This modified non-restoring division algorithm has been verified for a 45nm technology using Verilog models and simulations using Synopsys Verilog Compiler Simulator and Design Vision.

Chapter 5

Improved Non-restoring Square Root Algorithm

5.1 OVERVIEW

Today, the square root computation is one of the requirements for modern microprocessors although hardware designers take it as an infrequent, bothersome operation. So, they always take more time and effort on optimizing addition and multiplication, the most frequent operations rather than square root and division [21, 22]. As a result, addition and multiplication usually require from 2 to 5 machine cycles while division and square root latencies range from 13 to 60 cycles [22]. However, the square root and division operations are important since CAD tools and 3D graphic rendering applications, which use them frequently are popular. Also the IEEE floating-point standard specifies the square root operation as a basic arithmetic operation along with the 4 popular basic operations of add, subtract, multiply and divide [10].

Many square root computation algorithms have been proposed, for example, the pencil-and-paper algorithm, various digit recurrence square root algorithms including the restoring and non-restoring square root, square rooting by convergence with the Newton-Raphson method and table lookup. The digit recurrence square root using subtraction algorithm is the most popular since the square root convergence algorithm has a couple of drawbacks. The first is the serialized implementation that performs multiplication, division and square root operation in a single pipeline, which can lead to low throughput although the size can be minimized by sharing the multiplication unit [23, 28, 29]. The next one is it is hard to satisfy the IEEE standard on the accuracy of the final result, even though it has quadratic convergence [13, 21]. For table lookup, the output values are

already calculated and saved in a table so the computation is not necessary. It is very fast, but the size of lookup table can be huge for large wordsizes [24].

In this Chapter, the non-restoring square root algorithm, which is the fastest of the radix-2 digit recurrence square root algorithms [24] is discussed and examined for reducing the delay. In Section 5.2, the concept of the non-restoring square root algorithm is explained. Then, two approaches already presented in Chapters 3 and 4 are suitably modified for the non-restoring square root algorithm and they are introduced in Section 5.3. Finally, both analysis and simulation are performed for verification purposes in Section 5.4.

5.2 NON-RESTORING SQUARE ROOT ALGORITHM

The square-root algorithm can be processed through a series of shifts and adds similar to the division algorithm. The following notations are used in the Section.

z: Radicand	$z_1 z_0 . z_{-1} z_{-2} \ldots z_{-l}$	$(1 \leq z < 4)$
q: Square-root	$1 . q_{-1} q_{-2} \ldots q_{-l}$	$(1 \leq q < 2)$
s: Scaled remainder	$s_1 s_0 . s_{-1} s_{-2} \ldots s_{-l}$	$(0 \leq s < 4)$

The radicand for the non-restoring square root is in the range corresponding to the significand of a floating-point number in the IEEE standard format since the square root algorithm is practically applied to the floating-point numbers [14]. If the exponent is odd, it should be decreased by 1 to make the exponent even and the significand needs to shift to the left by 1. That's why the radicand has the extended range of $1 \leq z < 4$.

The non-restoring square root algorithm can be defined by these notations.

$$s^{(j)} = 2s^{(j-1)} - q_{-j} (2q^{(j-1)} + 2^{-j} q_{-j})$$

Where $q^{(j)}$ represents the root up to its $-j$ th digit and $s^{(j)}$ is also the scaled remainder up to the $-j$ th digit in the above equation. Below is the proof of the preceding square-rooting recurrence. By definition, these two equations below are always correct.

$$q^{(j)} = q^{(j-1)} + 2^{-j} q_{-j} \quad , \quad s^{(j)} = z - [\, q^{(j)} \,]^2$$

Then, the subtraction from the scaled remainder up to $-j+1$ by the scaled remainder up to $-j$ can be derived below.

$$\begin{aligned} s^{(j-1)} - s^{(j)} &= [\, q^{(j)} \,]^2 - [\, q^{(j-1)} \,]^2 = [\, q^{(j-1)} + 2^{-j} q_{-j} \,]^2 - [\, q^{(j)} \,]^2 \\ &= 2^{-j} q_{-j} [\, 2q^{(j-1)} + 2^{-j} q_{-j} \,] \end{aligned}$$

Multiplying both sides by 2^j and rearranging the terms and redefining the j th partial remainder to be $2^j s^{(j)}$ yields the recurrence equation.

$$s^{(j)} = 2s^{(j-1)} - q_{-j} (2q^{(j-1)} + 2^{-j} q_{-j})$$

The root digit in the non-restoring square root is selected from the set {+1, -1}, exactly the same set used for the quotient of non-restoring division and it must be converted from the set {+1, -1} to conventional binary number via on-the-fly digit conversion.

Using the above equation, the partial remainder can be obtained. If $q_{-j} = 1$, then

$$q_{-j}\,(\,2q^{(j-1)} + 2^{-j}q_{-j}\,) \quad = \quad 2q^{(j-1)} + 2^{-j}$$

and $2q^{(j-1)} + 2^{-j}$ should be subtracted from the current partial remainder. Otherwise,

$$q_{-j}\,(\,2q^{(j-1)} + 2^{-j}q_{-j}\,) \quad = \quad -\,[\,2q^{(j-1)} - 2^{-j}\,]$$

and $2q^{(j-1)} - 2^{-j}$ should be added to the current partial remainder. Figure 5.1 shows an example that illustrates how to find each root bit using the non- restoring square root algorithm.

$s^{(0)}$:	0 0 0.1 0 1 1 0 0 0 0	$q_0 = 0$, $q^{(0)} = 0$
$2s^{(0)}$:	0 0 1.0 1 1 0 0 0 0 0	$q_{-1} = 1$, $q^{(1)} = 0.1$
$-[2x(0.)+2^{-1}]$:	0 0 0.1 0 0 0 0 0 0 0	
$s^{(1)}$:	0 0 0.1 1 1 0 0 0 0 0	$q_{-2} = 1$, $q^{(2)} = 0.11$
$2s^{(1)}$:	0 0 1.1 1 0 0 0 0 0 0	
$-[2x(0.1)+2^{-2}]$:	0 0 1.0 1 0 0 0 0 0 0	
$s^{(2)}$:	0 0 0.1 0 0 0 0 0 0 0	$q_{-3} = 1$, $q^{(3)} = 0.111$
$2s^{(2)}$:	0 0 1.0 0 0 0 0 0 0 0	
$-[2x(0.11)+2^{-3}]$:	0 0 1.1 0 1 0 0 0 0 0	
$s^{(3)}$:	1 1 1.0 1 1 0 0 0 0 0	$q_{-4} = -1$, $q^{(3)} = 0.111\bar{1}$ $= 0.1101$
$2s^{(3)}$:	1 1 0.1 1 0 0 0 0 0 0	
$+[2x(0.1101)-2^{-4}]$:	0 0 1.1 0 1 1 0 0 0 0	
$s^{(4)}$:	0 0 0.0 1 1 1 0 0 0 0	$q_{-5} = 1$, $q^{(3)} = 0.11011$

Figure 5.1: Example of non-restoring square root process

5.3 THE IMPROVED NON-RESTORING SQUARE ROOT

Although the square root algorithm is very similar to the division algorithm, there are several different aspects to consider when the implementation is undertaken [25, 26, 27]. Figure 5.1 shows the implementation of the standard non-restoring square root algorithm for a 16-bit square root already discussed in Section 5.2. All the variables in the algorithm are 18-bit-width words although the input vector, the radicand, is 16-bit-width word. The partial remainder is produced by the equation,

$$s^{(j)} = 2s^{(j-1)} - q_{-j} (2q^{(j-1)} + 2^{-j} q_{-j})$$

and, it shows that not only the current partial remainder but also the current square root should be multiplied by 2 for finding the next partial remainder. Then, the addition or the subtraction process for the two operands is selected based on the sign bit of the current partial remainder. So, the additional 2 bits are required to check the sign bit of the next partial remainder since both the current partial remainder and the current square root are left shifted by 1 bit with the multiplication process. The 3 addition processes are also necessary per iteration and 2 additions are used for finding the next partial remainder $s^{(j)}$ using the above equation and for calculating $q^{(j-1)}$.

When the radicand (or the partial remainder) arrives at the multiplexer, the square root digits for non-restoring square root algorithm are selected from the set {+1, -1}, with +1 corresponding to the positive sign bit and subtraction and -1 to the negative sign bit. And, it also selects either $q^{(j-1)}$ or $-q^{(j-1)}$ as an operand for producing the next partial remainder. Once the operand is selected, the current partial remainder $s^{(j-1)}$ and the current square root $q^{(j-1)}$ are multiplied by 2. Then, 2^{-j} is subtracted from the doubled current

square root $2q^{(j-1)}$ or its complement $-2q^{(j-1)}$. Finally, the next partial remainder $s^{(j)}$ can be produced by adding the doubled current partial remainder $2s^{(j-1)}$ to $-q_{-j}(2q^{(j-1)} + 2^{-j}q_{-j})$.

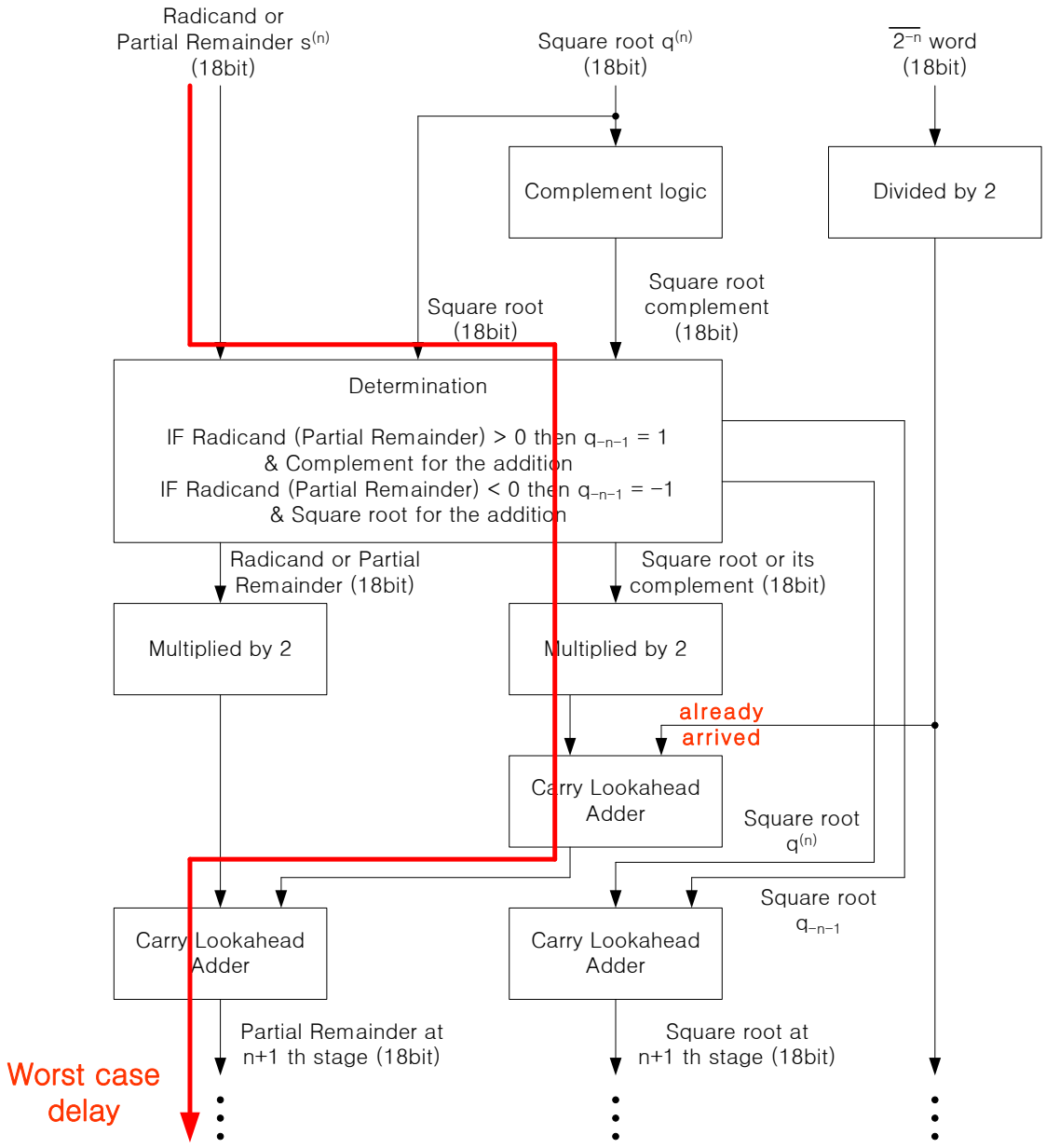


Figure 5.2: The flowchart for the 16-bit standard non-restoring square root algorithm

The standard non-restoring square root algorithm is a fully sequential process and the worst case delay comes from the path starting at the determination block and ending at the second carry lookahead adder. For the 16-bit square root algorithm, the worst case delay is the sum of the delays for the two adders and a multiplexer, 33 unit gate delays.

The modified non-restoring square root is presented in Figure 5.3 and this algorithm is adopted from the modified non-restoring division algorithm presented in Chapter 3 and suitably modified for the non-restoring square root algorithm. The order of the determination block and three carry lookahead adders is switched in order to follow the structure of the non-restoring division algorithm. However, it turns out that this structure cannot reduce the delay due to the existence of the complement logic which complements the current square root $q^{(n)}$. It is located between the determination block and the adders and increases the total delay by one unit gate delay. So, the one unit gate delay reduction caused from the early arrival of the radicand (or the partial remainder) at the multiplexer cancels out the one unit delay increase from the complement logic.

There are three ways to improve the square root algorithm using the ideas from the modified non-restoring division algorithm. The first, is that the first square root $q^{(1)}$, does not have to be generated since it is always set to one. Since the radicand entering into the floating-point square root is always positive [14], thus the process to find the MSB bit of the square root can be ignored. Second, the MSC generator can reduce the total delay and the area effectively as shown in Chapter 4. The MSC generator is used at the final stage for the modified non-restoring square root algorithm and its usage is extended to whole steps for the improved non-restoring square root algorithm to improve its performance. Finally, the square root generator can replace the adder for generating the current square root $q^{(j-1)}$ and it does not require an adder. Figure 5.4 shows how the

square root generator finds the current square root up to -*j*th digit. By this method, one adder can be easily removed from the algorithm.

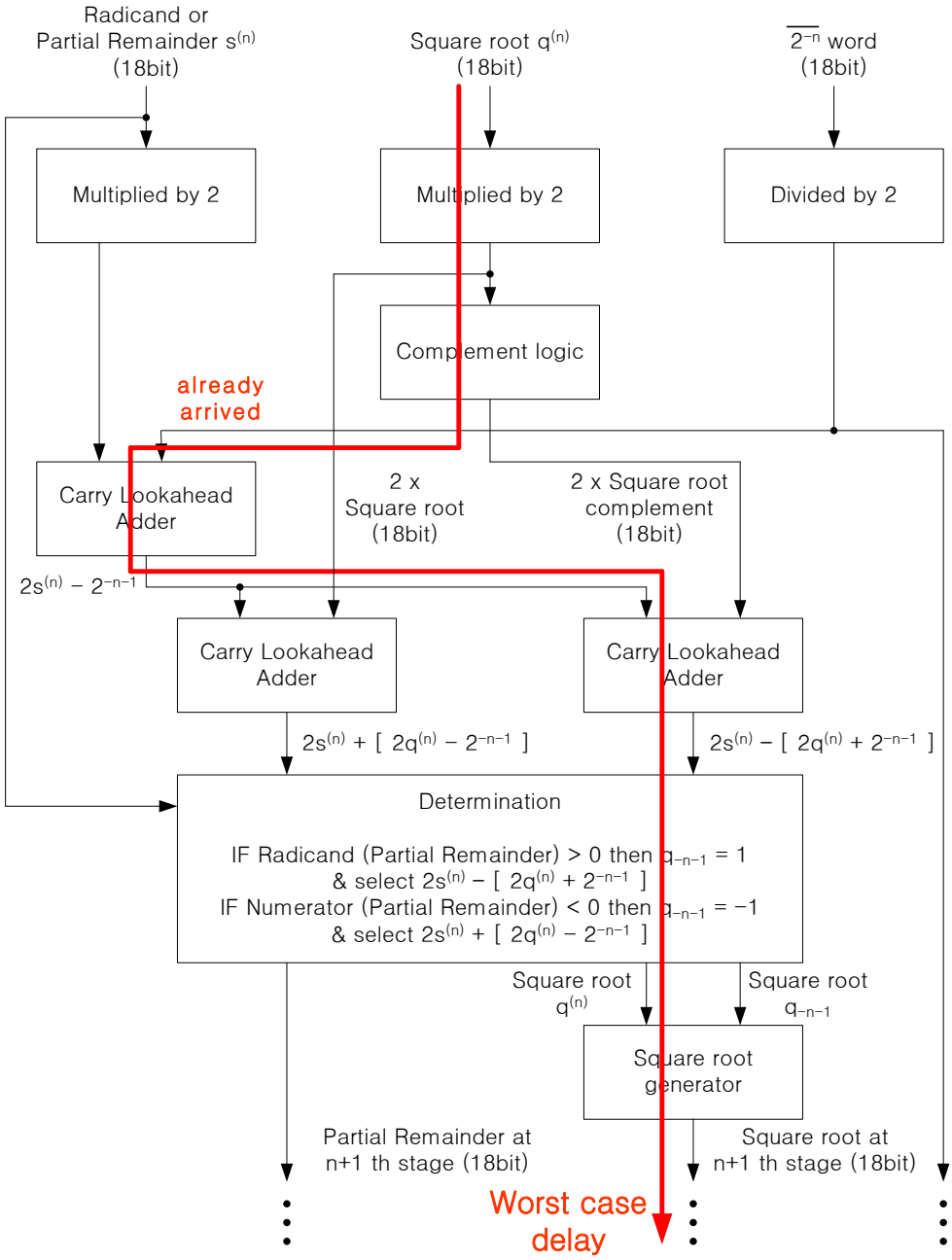


Figure 5.3: The flowchart for the 16-bit modified non-restoring square root algorithm

If $n>1$ & $q_{-n} = -1$ then, $q^{(n)} = 0.q_{-1}\, q_{-2}\, q_{-3} \cdots q_{-n+2}\, q_{-n+1}\, q_{-n}$

$= 0.q_{-1}\, q_{-2}\, q_{-3} \ldots q_{-n+2}\, 0\ \ 1$

If $n>1$ & $q_{-n} = +1$ then, $q^{(n)} = 0.q_{-1}\, q_{-2}\, q_{-3} \cdots q_{-n+2}\, q_{-n+1}\, q_{-n}$

$= 0.q_{-1}\, q_{-2}\, q_{-3} \ldots q_{-n+2}\, 1\ \ 1$

Ex) $q_0 = 0$, $q^{(0)} = 0$

$q_{-1} = 1$, $q^{(1)} = 0.1$

$q_{-2} = 1$, $q^{(2)} = 0.11$

$q_{-3} = -1$, $q^{(3)} = 0.11\overline{1} = 0.101$

$q_{-4} = 1$, $q^{(4)} = 0.1011$

$q_{-5} = -1$, $q^{(5)} = 0.1011\overline{1} = 0.10101$

Figure 5.4: The square root generator through on-the-fly calculation method

The improved non-restoring square root is presented in Figure 5.5. Like the non-restoring division method, it has a dual path that reduces the delay of the multiplexer. Since the worst case delay comes from the path with the two lookahead adders, it hides the delay of the adjacent path that includes the multiplexer and the MSC generator. For the 16-bit improved non-restoring square root algorithm, the sum of the delay from the series of 17 and 18-bit adders are 28 unit gate delays while the delay starting from the 18-bit multiplexer to the 18-bit modified MSC generator is only 12 unit gate delays. So, the delay from the adders is assumed to be the total delay for each iteration. Two adders have 24 unit gate delays and the combined multiplexer and the MSC generator has 11 unit gate delays per iteration for the 8-bit improved non-restoring square root algorithm and the delay from the adders is large enough to hide that of combined logic, too.

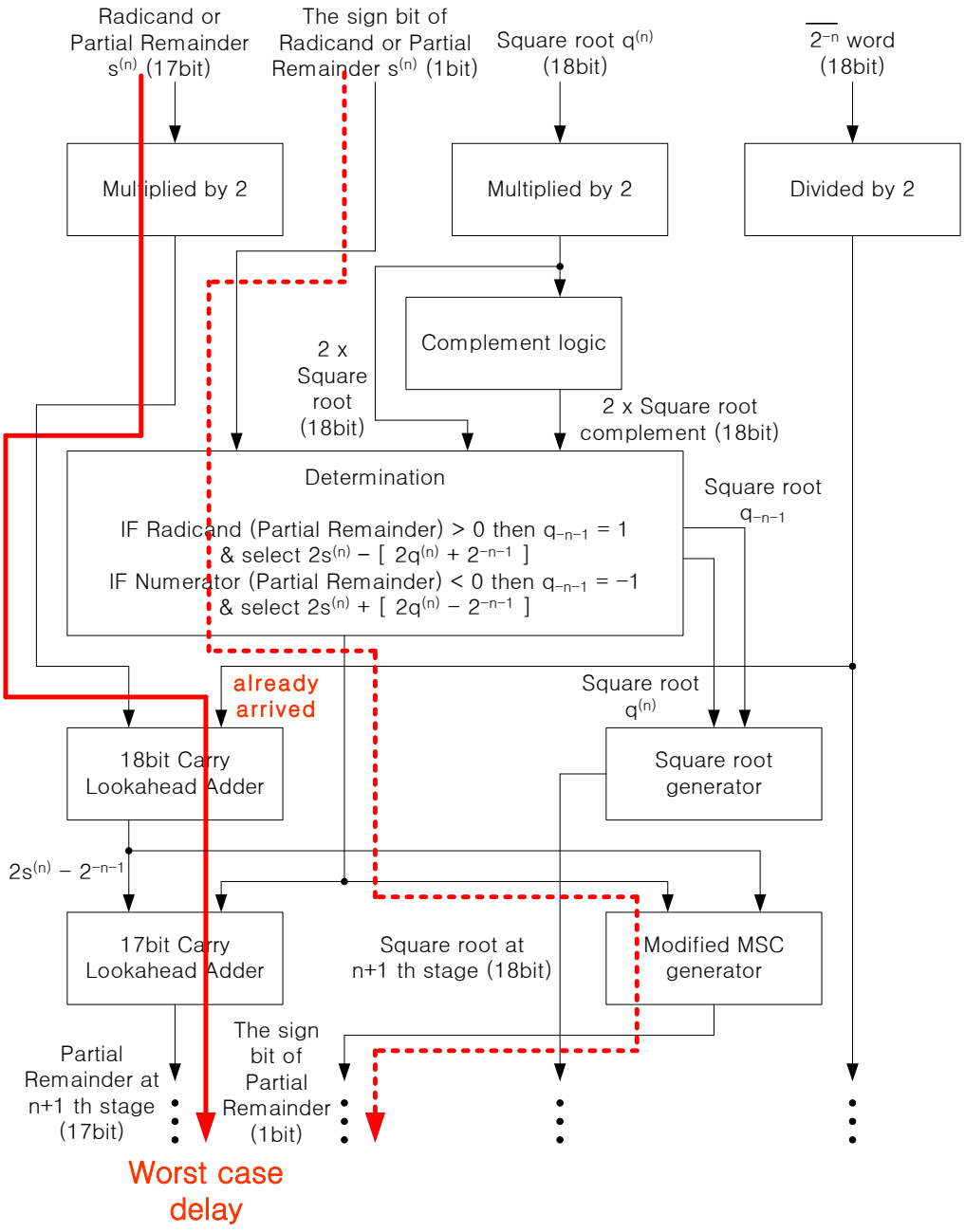


Figure 5.5: The flowchart for the 16-bit improved non-restoring square root algorithm

The same situation happens to the 32-bit improved non-restoring square root algorithm. The improved non-restoring square root algorithm also uses the first square root $q^{(1)}$ as algorithm simplification. It also removes one addition process per iteration by using the square root generator as shown in Figure 5.4. Thus two algorithms for the non-restoring square root with the approaches previously presented in Chapters 3 and 4 have been presented. The improved non-restoring square root algorithm successfully reduces the delay using dual path calculation while the modified non-restoring square root does not show a good result based on the flow chart analysis. The two algorithms for the non-restoring square root will be implemented and simulated in Section 5.4.

5.4 VERIFICATION AND SIMULATION RESULTS

In this section, both analysis and simulation are performed for verification purposes. Nine different square root circuits are implemented and analyzed in order to find the best algorithm of the three with respect to the estimated unit gate delays and their complexities. The simulations are performed using the Verilog Hardware Description Language (Verilog HDL). The FreePDK45nm version 1.0 library of standard cells is used to determine the delay, the area and the power consumption.

5.4.1 Algorithm Analysis

The estimated delays and complexities for 9 different non-restoring square root algorithms are presented in Tables 5.1, 5.2 and 5.3. The square root algorithms have three sections including the first, the intermediate and the last section since the first and the last section may have different logic and structure compared to the intermediate sections.

Table 5.1: The estimated unit gate delays and the complexities for the 3 different 8-bit square root algorithms

	8bit standard NRSR			8bit modified NRSR			8bit improved NRSR		
	Components	Delay	Complexity	Components	Delay	Complexity	Components	Delay	Complexity
1st stage	2 10bit inverter	1	20	10bit inverter	1	10	10bit inverter	1	10
	10bit MUX	2	31	10bit CLA	13	133	10bit CLA	13	133
	8bit CLA		113				10bit MSC generator		48
	2 10bit CLA	26	266						
Inter-mediate stage (2nd to 7th stage)	10bit inverter		60	10bit inverter	6	60	10bit inverter		60
	10bit MUX	18	186	3 10bit CLA	156	2394	10bit MUX		186
	8bit CLA		678	10bit MUX	12	186	10bit CLA	78	798
	2 10bit CLA	156	1596				9bit CLA	66	744
							10bit MSC generator		288
Final stage (8th stage)	10bit inverter		10	10bit inverter		10	10bit inverter		10
	10bit MUX	3	31	10bit CLA	13	133	10bit MUX		31
	8bit CLA		113	10bit MUX		31	10bit CLA	13	133
	2 10bit CLA	26	266	10bit MSC generator	8	48	10bit MSC generator	8	48
	1bit inverter	1	1						
Total	-	233	3371	-	209	3005	-	179	2489
Percen-tage (%)		100.0%	100.0%		89.7%	89.1%		76.8%	73.8%

Table 5.2: The estimated unit gate delays and the complexities for the 3 different 16-bit square root algorithms

	16bit standard NRSR			16bit modified NRSR			16bit improved NRSR		
	Components	Delay	Complexity	Components	Delay	Complexity	Components	Delay	Complexity
1st stage	2 18bit inverter	1	36	18bit inverter	1	18	18bit inverter	1	18
	18bit MUX	2	55	18bit CLA	15	258	18bit CLA	15	258
	16bit CLA		238				18bit MSC generator		90
	2 18bit CLA	30	516						
Inter-mediate stage (2nd to 15th stage)	18bit inverter		252	18bit inverter	14	252	18bit inverter		252
	18bit MUX	42	770	3 18bit CLA	420	10836	18bit MUX		770
	16bit CLA		3332	18bit MUX	28	770	18bit CLA	210	3612
	2 18bit CLA	420	7224				17bit CLA	182	3486
							18bit MSC generator		1260
Final stage (16th stage)	18bit inverter		18	18bit inverter		18	18bit inverter		18
	18bit MUX	3	55	18bit CLA	15	258	18bit MUX		55
	16bit CLA		238	18bit MUX		55	18bit CLA	15	258
	2 18bit CLA	30	516	18bit MSC generator	9	90	18bit MSC generator	9	90
	1bit inverter	1	1						
Total	-	529	13251	-	502	12555	-	432	10167
Percen-tage (%)		100.0%	100.0%		94.9%	94.7%		81.7%	76.7%

Table 5.3: The estimated unit gate delays and the complexities for the 3 different 32-bit square root algorithms

	32bit standard NRSR			32bit modified NRSR			32bit improved NRSR		
	Components	Delay	Complexity	Components	Delay	Complexity	Components	Delay	Complexity
1st stage	2 34bit inverter	1	68	34bit inverter	1	34	34bit inverter	1	34
	34bit MUX	2	103	34bit CLA	17	501	34bit CLA	17	501
	32bit CLA		481				34bit MSC generator		172
	2 34bit CLA	34	1002						
Inter-mediate stage (2nd to 31st stage)	34bit inverter		1020	34bit inverter	30	1020	34bit inverter		1020
	34bit MUX	90	3090	3 34bit CLA	1020	45090	34bit MUX		3090
	32bit CLA		14430	34bit MUX	60	3090	34bit CLA	510	15030
	2 34bit CLA	1020	30060				33bit CLA	510	14760
							34bit MSC generator		5160
Final stage (32nd stage)	34bit inverter		34	34bit inverter		34	34bit inverter		34
	34bit MUX	3	103	34bit CLA	17	501	34bit MUX		103
	32bit CLA		481	34bit MUX		103	34bit CLA	17	501
	2 34bit CLA	34	1002	34bit MSC generator	10	172	34bit MSC generator	10	172
	1bit inverter	1	1						
Total	-	1185	51875	-	1155	50545	-	1065	40577
Percen-tage (%)		100.0%	100.0%		97.5%	97.4%		89.9%	78.2%

At the first stage for the modified and the improved non-restoring square root algorithms, there are neither multiplexers nor adders for generating the first square root $q^{(1)}$ for the algorithm simplification since the radicand in the floating-point number is always greater than zero and the first square root $q^{(1)}$ is set to one. For the intermediate stages, the same processes are performed and repeated n-2 times to produce their square root using its own process. Three adders, a multiplexer and an inverter are used for the both the standard and the modified non-restoring square root algorithms in the intermediate stage and their delays per iteration are exactly the same. The area of the modified non-restoring square root algorithm is bigger than the standard non-restoring square root although one adder is replaced by the on-the-fly calculation unit. The improved non-restoring square root algorithms is the fastest and the smallest of the three different algorithms at the intermediate stages. Due to the dual path calculation, the delay of the multiplexer is eliminated. Its complexity is less than that of the other two algorithms because it uses the MSC generator instead of an adder. For example, only 2075 gates are used to implement the intermediate stages of the improved algorithm compared to 2520 and 2640 gates for the standard and the modified algorithms, respectively. Regarding the last stage, the standard non-restoring square root is the slowest and the most complex of the three since it needs to run an extra iteration compared to the other two algorithms. The other two algorithms use an MSC generator to reduce the delay and the complexity. Please note that either a 2-input AND or OR gate or an inverter is counted as 1 gate for the calculating the total complexity. From the delay perspective, either a 2-input AND or OR gate or an inverter has 1 unit gate delay.

First, 3 different 8-bit square root algorithms are analyzed as shown in Table 5.1. The improved non-restoring square root algorithm is the fastest and it has 23.2% less delay compared to the standard non-restoring square root algorithm. The delay of the

modified non-restoring square root algorithm is only 10.3% less than the standard algorithm and the reduction is mainly due to algorithm simplification, but the algorithm uses lots of adders. So, the reduction effects will fade away as the number of bits is increased. To sum up, the improved non-restoring square root algorithm reduces its delay by 23.2% and it also has 26.2% less area than the standard non-restoring square root algorithm. So, it has better overall performance than the modified non-restoring square root algorithm.

For the 16 and 32-bit square root algorithm comparison charts shown in Tables 5.2 and 5.3, the improved non-restoring square root algorithm is the best with respect to both the delay and the complexity. The total unit gate delays are reduced by 18.3% and 10.1% for 16 and 32-bit improved non-restoring square root, respectively while the modified non-restoring square root algorithm reduces its delays by 5.1% and 2.5% compared to the standard non-restoring square root. So, the modified non-restoring square root algorithm has little advantage over the standard non-restoring square root regarding the unit gate delay. The improved non-restoring division algorithm has 23.3% and 21.8% less complexity for 16 and 32-bit improved non-restoring square root, respectively. However, the complexity of the modified non-restoring square root algorithm is almost the same as that of the standard non-restoring square root algorithm since the difference between the two is only 2.6% for the 32-bit and 5.3% for the 16-bit modified algorithm.

The analysis shows the improved non-restoring square root algorithm reduces the delay by up to 23.2% and the complexity by up to 26.2%. And, the modified algorithm has little difference when comparing to the standard non-restoring square root

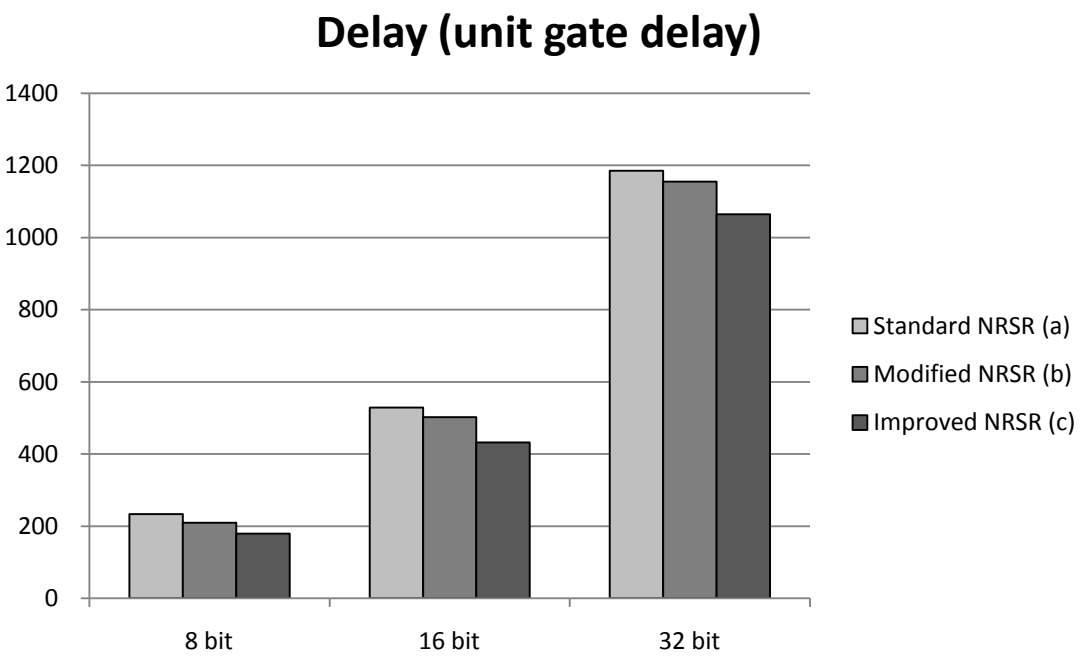


Figure 5.6: Graph of the estimated delays for each square root algorithm

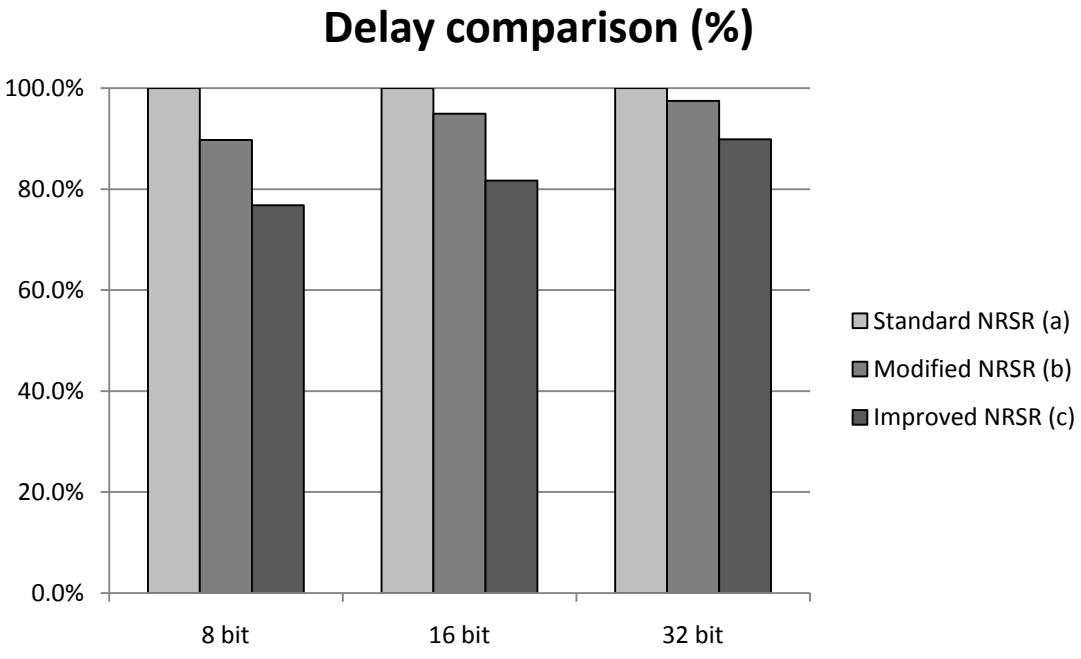


Figure 5.7: Graph of the estimated delay comparison between the three algorithms

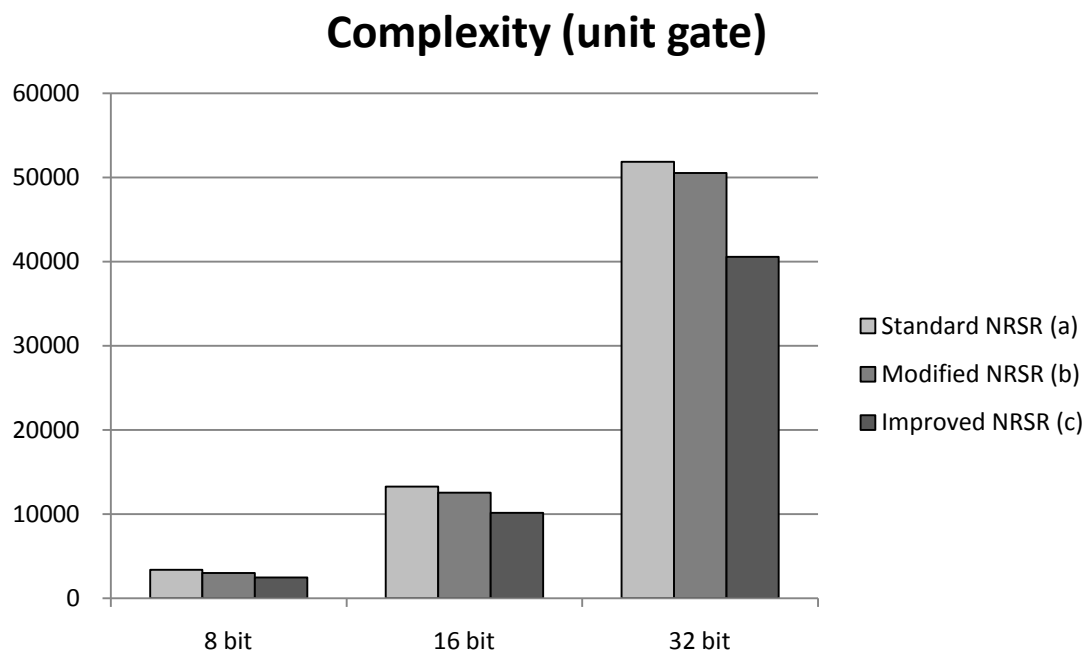


Figure 5.8: Graph of the complexities for each square root algorithm

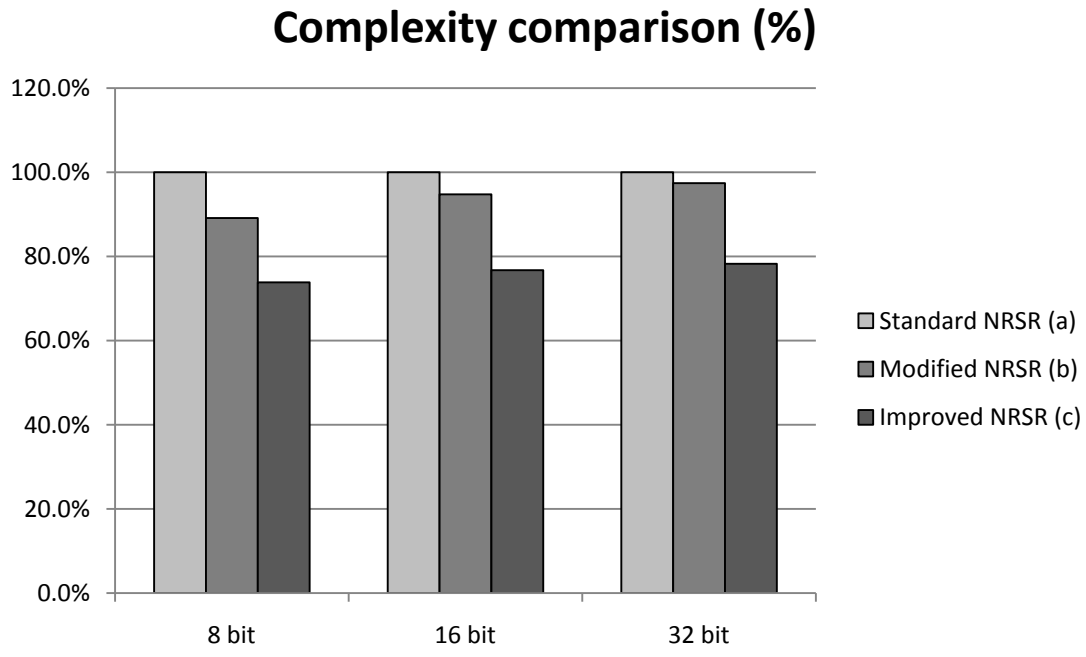


Figure 5.9: Graph of the complexity comparison between the three algorithms

division. This analysis is compared to the simulation results to find the non-restoring division algorithm with the best performance.

5.4.2 Simulation Results

Three different square root circuits based the standard, the modified and the improved non-restoring square root algorithms are implemented changing their bit width from 8 to 32-bit using the Verilog Hardware Description Language (Verilog HDL). These Verilog models have been synthesized using the Synopsys Verilog Compiler Simulator (VCS) and the FreePDK45nm version 1.0 is used as a library file for the delay, the area and the power consumption. Design Vision is also used for finding the area and the total power consumption. A hundred random vectors are generated as both the numerators and the denominators for each simulation and the delays in Table 5.4 are the average values of a hundred simulations.

Table 5.4: Average delays for each square root algorithm

delay	Standard NRSR (a)	Modified NRSR (b)	Improved NRSR (c)	Diff. (a-b)	Diff. (a-c)
8 bit	28.00 ns	26.93 ns	22.54 ns	1.07 ns	5.46 ns
16 bit	60.40 ns	59.95 ns	52.59 ns	0.45 ns	7.81 ns
32 bit	126.74 ns	128.20 ns	116.55 ns	-1.46 ns	10.19 ns

Table 5.4 shows that the improved non-restoring square root algorithm is the fastest compared to both the standard and the modified non-restoring square root algorithm. For the 8-bit comparison, the improved non-restoring square root algorithm has a delay of 22. 54 nanoseconds, which is 19.5% less than the standard. For 16 and 32

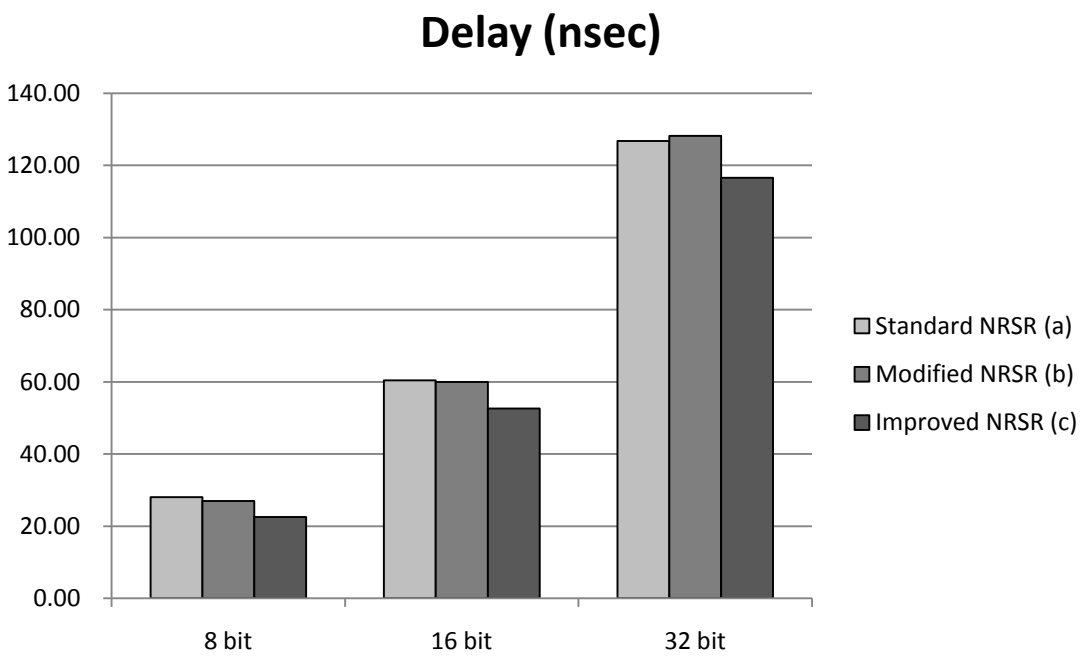


Figure 5.10: Graph of the delays for each square root algorithm

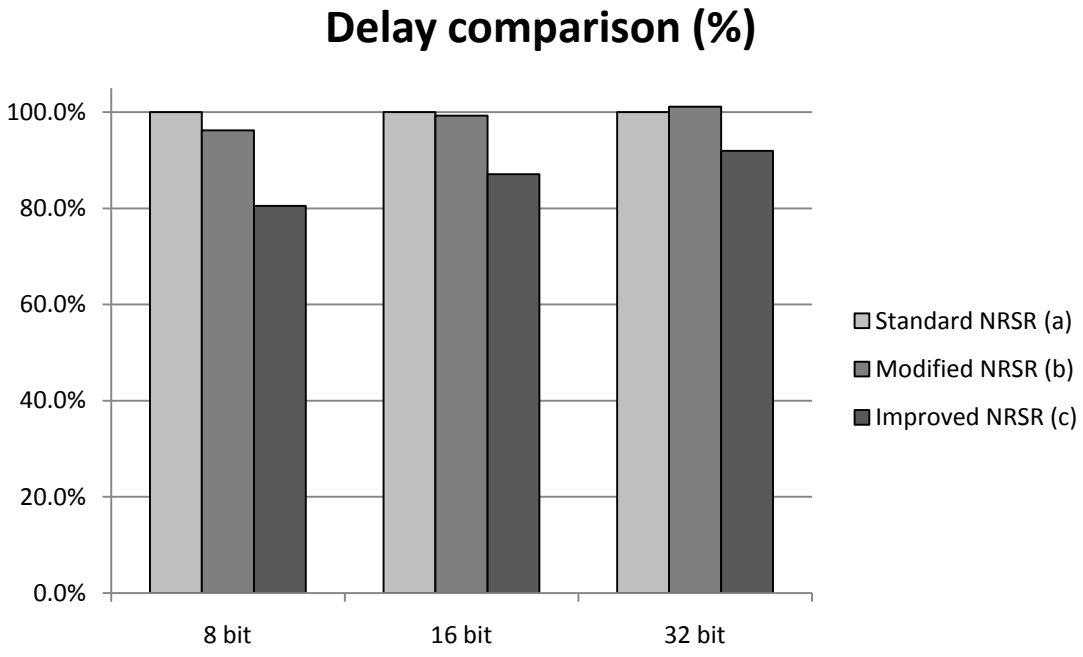


Figure 5.11: Graph of the delay comparison between the three algorithms

bit comparison, the improved algorithm has delays of 52.59 and 116.55 nanoseconds, which are 12.9% and 8.0% less than the standard algorithm. Although the amount of delay in time is increased as the number of bits is increased, the rate of the delay reduction, on the contrary, is decreased for both the modified and the improved non-restoring square root algorithm. And, the simulation results almost coincide with the analysis as shown in Section 5.4.1. For the modified non-restoring square root, the result shows it has almost the same delay as the standard or even slightly slower than the standard for the 32-bit comparison.

Table 5.5: Areas for each square root algorithm

Area	Standard NRSR (a)	Modified NRSR (b)	Improved NRSR (c)	Diff. (a-b)	Diff. (a-c)
8 bit	$7727\,\mu m^2$	$6511\,\mu m^2$	$5352\,\mu m^2$	$1216\ \mu m^2$	$2375\ \mu m^2$
16 bit	$29818\,\mu m^2$	$27302\,\mu m^2$	$22024\,\mu m^2$	$2516\ \mu m^2$	$7794\ \mu m^2$
32 bit	$116456\,\mu m^2$	$110831\,\mu m^2$	$88827\,\mu m^2$	$5625\ \mu m^2$	$27629\ \mu m^2$

Both the modified and improved square root algorithms have smaller area than the standard square root algorithm since they employ the square root generator which has few logic gates inside and it reduces the areas considerably. This advantage is almost useless for the modified non-restoring square root algorithm since it uses one more adder per iteration, which is enough to cancel out the advantage of using the square root generator. In Table 5.5, the modified algorithm has areas of $6511\,\mu m^2$, $27302\,\mu m^2$ and $110831\,\mu m^2$, which are 15.7%, 8.4% and 4.8% less than the standard algorithm respectively. For the improved algorithms, it has areas of $5352\,\mu m^2$, $22024\,\mu m^2$ and 88827

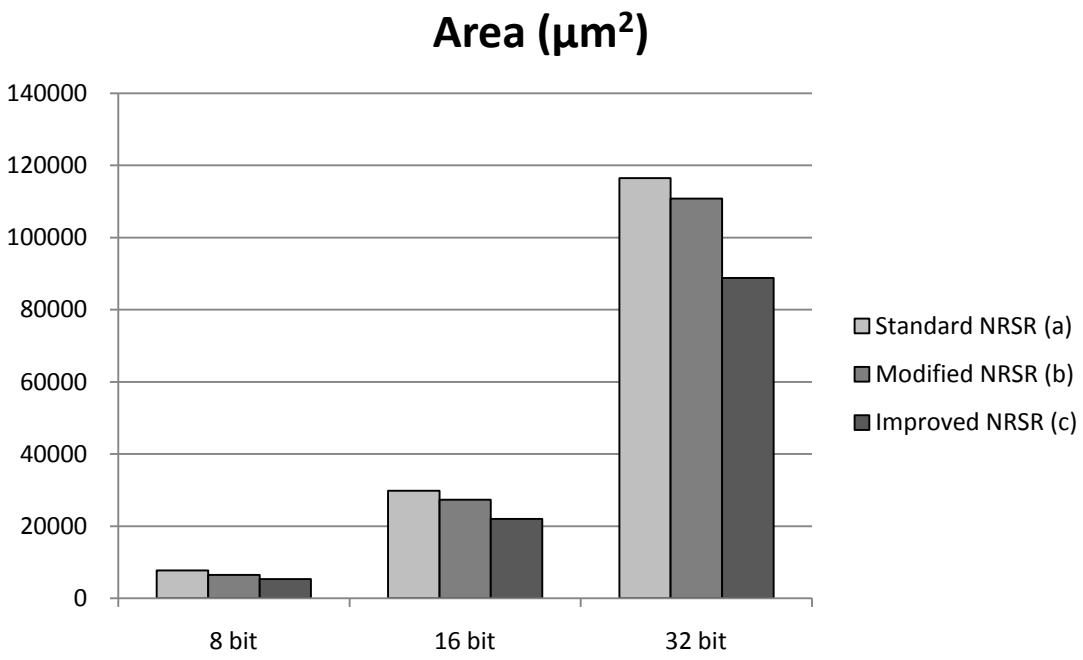


Figure 5.12: Graph of the areas for each square root algorithm

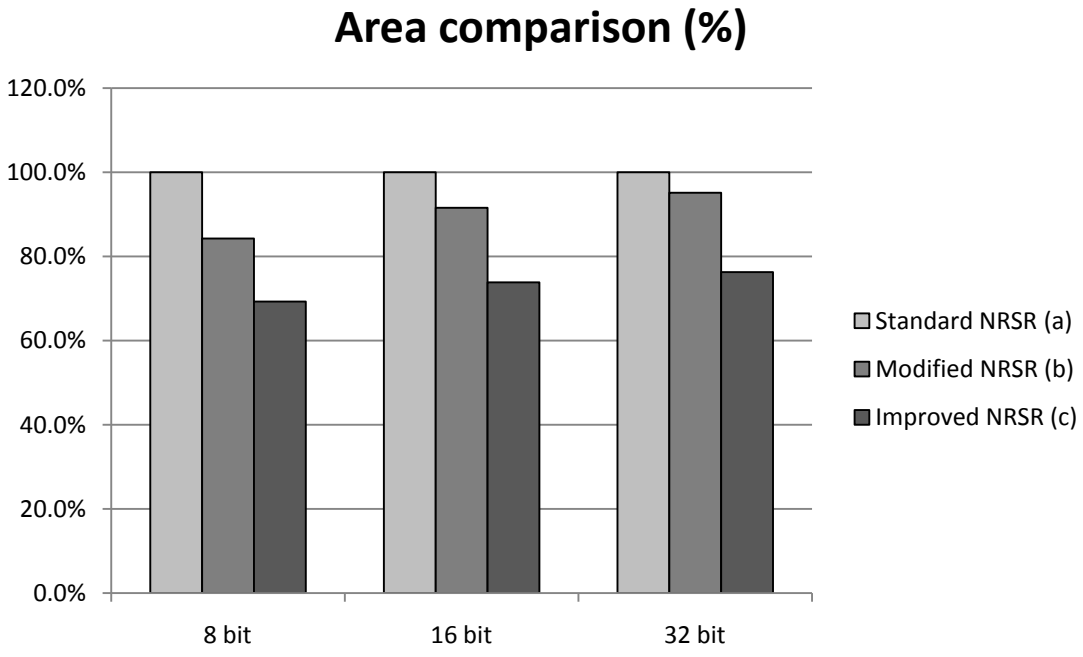


Figure 5.13: Graph of the area comparison between the three algorithms

μm^2, which are 30.6% , 26.1% and 23.7% less than the standard algorithm respectively. From analysis of the complexities for each square root algorithm, the complexities are decreased by 26.2%, 23.3% and 21.8%, respectively for the 8, 16 and 32-bit improved non-restoring division algorithms when compared to the standard non-restoring division algorithms. So, both the simulated and the analyzed results are very similar with respect to the areas. The improved non-restoring division algorithm has less area than other two algorithms.

Table 5.6: Total power consumption for each square root algorithm

Power	Standard NRSR (a)	Modified NRSR (b)	Improved NRSR (c)	Diff. (a-b)	Diff. (a-c)
8 bit	1076.2 µW	945.1 µW	624.9 µW	131.1 µW	451.3 µW
16 bit	3603.1 µW	3194.9 µW	2647.4 µW	408.2 µW	1448.3 µW
32 bit	10885.7 µW	10185.0 µW	7427.6 µW	700.7 µW	3458.1 µW

Table 5.6 shows the total power consumption for each square root algorithm. The modified square root algorithm has power consumptions of 945.1 µW, 3194.9 µW and 10185.0 µW, which are 12.2%, 11.3% and 6.4% less than the standard algorithm, respectively. In Table 5.5, it has areas of 6511 μm^2, 27302 μm^2 and 110831 μm^2, which are 15.7%, 8.4% and 4.8% less than the standard algorithm respectively. So, it is true that the power consumption increases as the area increases. The improved square root algorithm has the least power consumption for 8, 16 and 32 bit square root circuits since it has less area than the other square root circuits. To sum up, the improved non-restoring square root algorithm shows the best performance of the three non-restoring square roots.

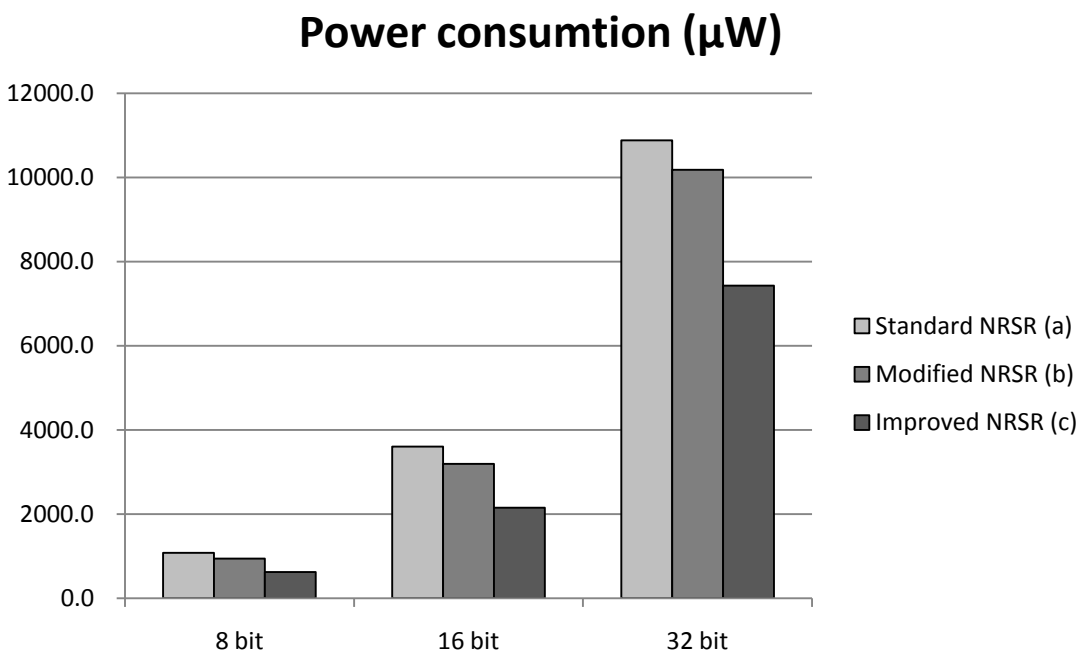


Figure 5.14: Graph of the power consumption for each division algorithm

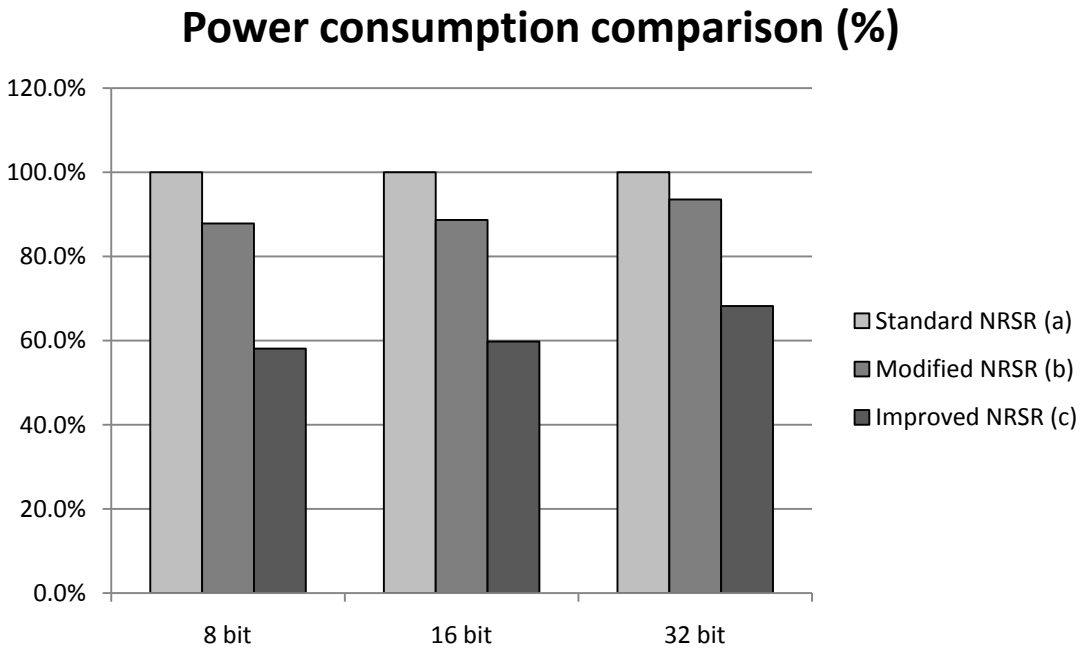


Figure 5.15: Graph of the power consumption comparison between the three algorithms

5.5 SUMMARY

Two new algorithms for the non-restoring square root that adopts the concepts already applied for the non-restoring division are presented. The improved non-restoring square root algorithm with dual path calculation increases its speed and minimizes the area and the power consumption compared to the modified non-restoring division algorithm. The main contribution of this proposed method is the dual path structure that enables the square root to perform two different processes simultaneously and that is also made possible by the modified MSC generator. It has been verified that the improved non-restoring square root reduces the total delay by up to 19.5% compared to the standard non-restoring square root. The improved non-restoring square root algorithm has almost 30.6% less area and up to 41.9% less power consumption than the standard non-restoring square root algorithm. This modified non-restoring square root algorithm has been verified for a 45nm technology using Verilog models and simulations using Synopsys Verilog Compiler Simulator and Design Vision.

Chapter 6

Conclusion and Future Work

6.1 CONCLUSIONS

This dissertation presents new algorithms for the non-restoring division and square root. To reduce the delay of non-restoring division, change to the order of the flowchart that reduces one unit delay per iteration is presented. To enhance the overall performance, the dual path calculation algorithm with the modified MSC generator is also proposed. Two concepts already applied for non-restoring division mentioned above are adopted for improving the performance of non-restoring square root.

A modified algorithm for non-restoring division has been presented. The new algorithm changes the order of the flowchart, which reduces one unit delay of the multiplexer per iteration so that it increases the speed. In addition, a new method to find a correct quotient using the MSC generator is presented and it removes an error that the quotient is always odd number after a digit conversion from a digit converter from the quotient with digits 1 and -1 to a conventional binary number. It has been verified that the modified non-restoring division reduces the total delay by almost 21% compared to the standard non-restoring division. The reduction increases as the word size increases. The least significant bit of the quotient is correctly generated via the most significant carry generator. The total area is increased by over 78% and the total power consumption is also increased by over 100% as a tradeoff.

The next research focuses on improving the delay profile for the non-restoring division algorithm without doubling the number of the adders producing the intermediate

partial remainders. The improved non-restoring division algorithm increases its speed and minimizes the area and the power consumption compared to the modified non-restoring division algorithm. The main contribution of this proposed method is the dual path structure that enables the divider to perform two different processes simultaneously and that is also made possible by the modified MSC generator. It has been verified that the improved non-restoring division reduces the total delay by up to 24% compared to the standard non-restoring division. The improved non-restoring division algorithm has almost 32% less area and up to 27% less power consumption than the modified non-restoring division although it still has a larger area and higher power consumption than the standard method.

Two new algorithms for the non-restoring square root that adopt the concepts already applied for the non-restoring division are discussed. The improved non-restoring square root algorithm with dual path calculation increases its speed and minimizes the area and the power consumption compared to the modified non-restoring division algorithm. It has been verified that the improved non-restoring square root reduces the total delay by up to 19.5% compared to the standard non-restoring square root algorithm. The improved non-restoring square root algorithm has almost 30.6% less area and up to 41.9% less power consumption than the standard non-restoring square root algorithm.

6.2 PUBLISHED RESULTS

This research has resulted in conference paper [30]. Another conference paper has been submitted [31].

6.3 FUTURE WORK

The proposed methods in this dissertation improves the delay while minimizing the area and the power consumption. Future work should focus on the non-restoring square root algorithms. Since a non-restoring square root has similar process to that of non-restoring division, it is hard to implement the new algorithm by adopting the concept already applied for the non-restoring division. For example, the modified non-restoring square root did not improve at all using the method applied for the modified non-restoring division.

The research also focused mainly on the delay and area reduction. So, the additional work should be done to find effective method to reduce the power consumption while maintaining the fast delay profile. Although the research on the arithmetic operations has been done since the presence of the mankind, there is a lots of work yet to be done for implementing efficient computation methods.

Bibliography

[1] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, vol. 46, pp. 154–161, 1997.

[2] Inwook Kong, "Modified Improved Algorithms and Hardware Designs for Division by Convergence," *Doctoral dissertation, University of Texas at Austin*, 2009.

[3] Peter Soderquist and Miriam Leeser, "Division and square root: choosing the right implementation," *IEEE Micro*, vol. 17, no. 4, pp. 56-66, July-Aug. 1997.

[4] Milos D. Ercegovac and Tomas Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Boston: Kluwer Academic Publishers, 1994.

[5] S. F. Oberman and M. J. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers*, vol. 46, pp. 833–854, 1997.

[6] S. F. Oberman, "Floating-point division and square root algorithms and implementation in the AMD-K7 microprocessor," *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pp. 106–115, 1999.

[7] Gustavo Sutter, Jean-Pierre Deschamps, Gery Bioul and Eduardo Boemo, "Power Aware Dividers in FPGA," *Power and Timing Modeling, Optimization and Simulation 2004*, LNCS 3254, pp. 574–584, 2004.

[8] Nikolay Sorokin, "Implementation of high-speed fixed-point dividers on FPGA," *Journal of Computer Science and Technology*, Vol. 6, No. 1, pp.8-11, 2006.

[9] J. E. Robertson, "A new class of digital division methods," *IRE Transactions on Electronic Computers*, vol. EC-7, pp. 218-222, Sept. 1958.

[10] Behrooz Parhami, *Computer Arithmetic - Algorithms and Hardware Designs*, Oxford: Oxford University Press, 2000.

[11] O. Bedrij, "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 340-346, 1962.

[12] J. D. Bruguera and Tomas Lang, "Multilevel Reverse Carry Adder," *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp. 155-162, 2000.

[13] Fandrianto, Jan, "Algorithm for high speed shared radix 4 division and radix 4 square-root, " *Computer Arithmetic (ARITH), 1987 IEEE 8th Symposium on*, pp.73-79, May 1987.

[14] ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, 1985.

[15] Kihwan Jun, "Modified Non-restoring Division Algorithm with Improved Delay Profile," *Master's thesis, University of Texas at Austin*, 2011.

[16] A. W. Burks, H. H. Goldstein, and J. von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, 2nd edition, Section 5.14, Princeton: Institute for Advanced Study, 1947.

[17] J. D. Bruguera and Tomas Lang, "Multilevel Reverse Most-significant Carry Computation," *IEEE Transactions on VLSI Systems*, vol.9, no.6, pp.959-962, Dec. 2001.

[18] Tomas Lang and J. D. Bruguera, "Multilevel reverse-carry computation for comparison and for sign and overflow detection in addition," *1999 International Conference on Computer Design*, pp.73-79, 1999.

[19] J. D. Bruguera and Tomas Lang, "Using the reverse-carry approach for double datapath floating-point addition, "*, Proceedings of the 2001 15th IEEE Symposium on Computer Arithmetic*, pp.203-210, 2001.

[20] D. Piso and J. D. Bruguera, "A new rounding algorithm for variable latency division and square root implementations," *11th Euromicro Symposium on Digital Systems Design*, pp. 760–767, 2008.

[21] Taek-Jun Kwon, Jeff Sondeen and Jeff Draper, "Floating-point division and square root implementation using a Taylor-series expansion algorithm, ", *15th IEEE International Conference on Electronics, Circuits and Systems,* pp.702-705, 2008.

[22] Tole Sutikno, Aiman Zakwan Jidin, Auzani Jidin and Nik Rumzi Nik Idris, "Simplified VHDL Coding of Modified Non-Restoring Square Root Calculator, ", *International Journal of Reconfigurable and Embedded Systems,* Vol.1, No.1, pp.37~42, 2012.

[23] Peter Soderquist and Miriam Leeser, "Area and performance tradeoffs in floating-point divide and square-root implementations," *ACM Computing Surveys*, vol. 28, no. 3, pp. 518-564, Sept. 1996.

[24] M. Franke, A. Th. Schwarzbacher, M. Brutscheck and St. Becker, "Implementation of different square root algorithms," *Proceedings of 6th Electronic Circuits and Systems Conference*, Bratislava, Slovakia, pp.103-106, Sep. 2007.

[25] Stanislaw Majerski, "Square-Rooting Algorithms for High-Speed Digital Circuits," *IEEE Transactions on Computers*, vol.C-34, no.8, pp.724-733, Aug. 1985.

[26] Yamin Li and Wanming Chu, "A New Non-Restoring Square Root Algorithm and Its VLSI Implementations," *ICCD 1996, International Conference on Computer Design*, pp.538-544, Oct. 1996.

[27] Milos D. Ercegovac and Tomas Lang, *Digital Arithmetic*, San Francisco: Morgan Kaufmann Publishers, 2004.

[28] I. Kong and E. E. Swartzlander, Jr., "A rounding method with improved error tolerance for division by convergence," *Proceedings of the 42nd Asilomar Conference on Signals, Systems and Computers*, pp. 1814–1818, 2008.

[29] M. D. Ercegovac, L. Imbert, D. W. Matula, J. M. Muller, and G. Wei, "Improving Goldschmidt division, square root, and square root reciprocal," *IEEE Transactions on Computers*, vol. 49, pp. 759–763, 2000.

[30] Kihwan Jun and Earl E. Swartzlander, Jr., "Modified Non-restoring Division Algorithm with Improved Delay Profile and Error Correction," *2012 46th Asilomar Conference on Signals, Systems and Computers,* accepted, 4-7 Nov. 2012.

[31] Kihwan Jun and Earl E. Swartzlander, Jr., "Improved Non-restoring Division Algorithm with dual path calculation," *The 38th International Conference on Acoustics, Speech, and Signal Processing*, submitted, Nov. 2012.

Vita

Kihwan Jun received the degree of Bachelor of Science in Radio Science and Engineering from Hanyang University, Seoul, Korea, in February, 1996. He received the degree of Master of Science in Electronic Communication Engineering with a thesis on analysis of the radio wave propagation characteristics using ray tracing method from the same university in February, 1998. During the following six and half years he has worked in passenger car development center of Hyundai Motor Company, LTD., Korea, where he developed automotive electronic systems for Hyundai Genesis and Equus. In August, 2004, he joined at the University of Texas at Austin where he started his graduate studies for another Master's and Ph.D. degree in the Electrical and Computer Engineering. He joined the application specific processor group under Dr. Swartzlander's supervising in April, 2006. He received the degree of Master of Science in Electrical Engineering with a thesis on modified non-restoring division algorithm with improved delay profile in May, 2011. His research interests are high-speed computer arithmetic algorithms, VLSI circuit designs and application specific processor designs.

Permanent email address: kihwan.jun@utexas.edu

This dissertation was typed by the author.